МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ, МОЛОДЕЖИ И СПОРТА УКРАИНЫ
ХАРЬКОВСКАЯ НАЦИОНАЛЬНАЯ АКАДЕМИЯ ГОРОДСКОГО ХОЗЯЙСТВА

С.М. Мордовцев

КОНСПЕКТ ЛЕКЦИЙ

по курсу **«Системы управления базами данных»**

(для студентов 3 – 4 курсов заочной формы обучения
направления подготовки 6.030601 "Менеджмент")

Харьков – ХНАГХ – 2012

Мордовцев С.М. Конспект лекций по курсу «Системы управления базами данных» (для студентов 3 –4 курсов заочной формы обучения направления подготовки 6.030601 "Менеджмент") / С.М. Мордовцев; Харьк. нац. акад. гор. хоз-ва. – Х.: ХНАГХ, 2012. - 62 с.

Автор: С.М. Мордовцев

Рецензент: к.т.н, доц.. И.Т. Карпалюк

Рекомендовано кафедрой Информационных систем и технологий в городском хозяйстве, протокол № 67 от 03.09.2010 г.

Содержание

ВВЕДЕНИЕ

Дисциплина «СУБД» имеет целью познакомить будущего менеджера с современной теорией баз данных, теорией и практикой их проектирования, с современными тенденциями развития систем управления базами данных.

В результате изучения дисциплины студент должен знать:

- принципы современной организации баз данных и систем баз данных;
- основные категории и понятия баз данных;
- реляционную модель представления данных;
- методы проектирования баз данных;
- современные технологии обработки данных;

В результате изучения дисциплины студент должен уметь:

строить модель предметной области и создать соответствующую ей базу данных;

- организовать обработку информации в базе данных;
- организовать обеспечение целостности базы данных;
- формулировать запросы к базе данных;
- организовать защиту базы данных.

В результате изучения дисциплины студент должен владеть:

· навыками работы в конкретной системе управления базами данных, научиться создавать основные объекты базы данных (таблицы, формы, отчеты);

· навыками реализации основных функций, необходимых для решения поставленной задачи;

· навыками создания приложений для работы с базой данных;

· навыками работы с распределенной базой данных;

· информационными технологиями, используемыми при решении реальных экономических задач.

С.М.1.1. Модели баз данных. Таблицы баз данных
Тема 1. Базы данных и СУБД

Основные вопросы
1.1. Понятие базы данных и системы баз данных
1.2. Компоненты системы баз данных
1.3. Администрация базы данных
1.4. Классификация баз данных
1.5. Классификация СУБД

1.1. Понятие базы данных и системы баз данных

С начала развития вычислительной техники образовались два основных направления ее использования:

- выполнение расчетов, которые невозможно производить вручную;
- создание автоматизированных информационных систем (АИС).

Создание АИС стало возможным с появлением жестких дисков большой емкости, обеспечивающих произвольный доступ к данным. Это предопределило развитие АИС разного назначения и масштаба, в первую очередь в области бизнес-приложений. Примерами являются автоматизированные системы управления предприятием, банковские системы, системы резервирования и продажи билетов.

Параллельно развивались системы управления физическими экспериментами, обеспечивающие оперативную обработку в реальном времени огромных потоков данных от датчиков, и автоматизированные библиотечные информационно-поисковые системы. Все это привело к появлению новой информационной технологии интегрированного хранения и обработки данных – концепции баз данных.

Система баз данных (СБД) – это система специально организованных данных (баз данных), программных, технических, языковых, организационно-методических средств для централизованного накопления и коллективного многоцелевого использования данных.

Терминологическое единство в рассматриваемой сфере отсутствует. Термин «система баз данных» (database system) широко используется в современной англоязычной литературе для определения человеко-машинной

системы, включающей БД, СУБД, оборудование и персонал. Значительно реже используется термин «банк данных», который многими авторами признается архаичным.

База данных (БД) – именованная совокупность данных, отображающая состояние объектов и их отношений в рассматриваемой предметной области

В ранних определениях БД указывалось на отсутствие дублирования данных. Однако дублирование может быть вызвано спецификой модели данных или технологическими причинами (обеспечение надежности, сокращение времени реакции). Но это должно быть отслеживаемое и управляемое дублирование.

Система управления базами данных (СУБД) – это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Иногда в составе СБД выделяют *систему управления архивами.* Под оперативным управлением СУБД находится часть данных, остальные данные (архивы) располагаются на носителях, не управляемых СУБД.

Основные требования к СБД можно сформулировать следующим образом:

- ·адекватность отображения предметной области (полнота, целостность, непротиворечивость и актуальность данных);
- ·возможность взаимодействия пользователей разных категорий, обеспечение высокой эффективности доступа;
- · дружественность интерфейса;
- ·обеспечение секретности и конфиденциальности;
- · обеспечение взаимной независимости программ и данных;
- ·обеспечение надежности – защита данных от случайного и преднамеренного разрушения, возможность восстановления данных в случае сбоев в системе;
- ·распределенная обработка данных и обеспечение эффективного доступа пользователей к данным в любой точке сети.

1.2. Компоненты системы баз данных

СБД является сложной человеко-машинной системой, включающей различные взаимосвязанные и взаимозависимые компоненты (подсистемы) (рис. 1.1).

Рис. 1.1. Компоненты СБД

Данные, отражающие состояние предметной области и используемые АИС, принято называть *информационной базой.* Информационная база включает:

- собственно данные;
- метаданные (описания этих данных).

Данные отделены от описаний, но в то же время данные не могут использоваться без обращения к соответствующим описаниям.

Языковые средства СУБД являются важнейшим компонентом СБД, так как обеспечивают интерфейс пользователей разных категорий с СБД (рис. 1.2). Языковые средства современных СУБД относятся к *4-му поколению*.

На рис. 1.3 приведены компоненты языка 4-го поколения. К 1-му поколению относят машинные языки, ко 2-му – языки ассемблера, к 3-му – алгоритмические языки типа PL и Cobol, которые назывались языками высокого уровня, но уровень которых гораздо ниже, чем у языков 4-го поколения. К 5-му поколению относят языки систем искусственного интеллекта (Prolog).

Для выражения обобщенного взгляда на данные применяют *язык*

описания данных (ЯОД), включаемый в состав СУБД. ЯОД позволяет определять схемы БД, характеристики хранимых данных, параметры хранения их в памяти и может включать средства поддержки целостности, ограничения доступа, секретности. Одна БД на ЯОД разных СУБД может описываться по-разному.

Рис. 1.2. Классификация языковых средств СБД

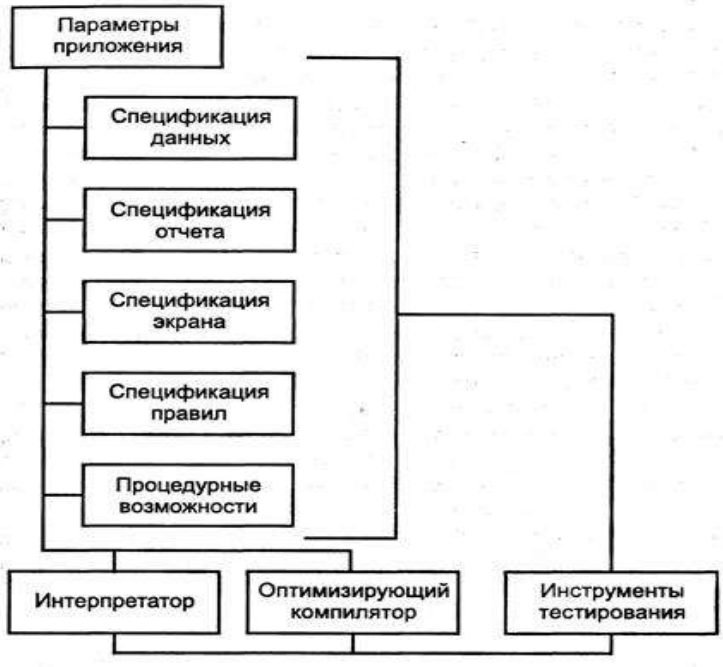

Рис. 1.3. Компоненты языка 4-го поколения

Язык манипулирования данными (ЯМД) включает в себя средства запросов к БД и поддержания БД (добавление, удаление, обновление данных, создание и уничтожение БД, обеспечение запросов к справочнику БД). ЯМД разделяются

- на процедурные;
- непроцедурные (декларативные).

При пользовании *процедурными языками* надо указать, какие действия и над какими объектами необходимо выполнить, чтобы получить результат. В *непроцедурных языках* указывается, что надо получить в ответе, а не как этого достичь. Процедурные языки могут различаться по основным информационным единицам, которыми они манипулируют. Это могут быть:

- языки, ориентированные на позаписную обработку данных;
- языки, ориентированные на операции над множеством записей.

Примерами непроцедурных языков являются языки, основанные на реляционном исчислении. К ним относятся язык запросов SQL и табличный язык QBE.

По форме представления различают следующие языковые средства:

- аналитические;
- табличные;
- графические.

В рамках одной СУБД могут использоваться языки разных типов. Во многих СУБД (dBase, FoxPro и др.) для манипулирования данными могут использоваться:

- табличный язык запросов типа QBE;
- аналитический язык запросов SQL;
- процедурный язык программирования (для dBase и FoxPro – язык xBase).

Кроме упомянутых языковых средств эти системы включают генераторы экранных форм, отчетов и приложений, а также язык разветвленной иерархической системы «меню», позволяющей пользователю выбрать нужные действия.

Наиболее распространенным языком является SQL (Structured Query Language), предоставляющий средства обработки запросов и функции по созданию, обновлению и управлению доступом. SQL соединяет в себе ЯОД и ЯМД. Он не является полноценным языком программирования. Для доступа к БД из прикладных программ SQL-выражения встраиваются в конструкции базового языка.

Программные средства СБД представляют собой сложный комплекс, обеспечивающий взаимодействие всех частей системы (рис. 1.4).

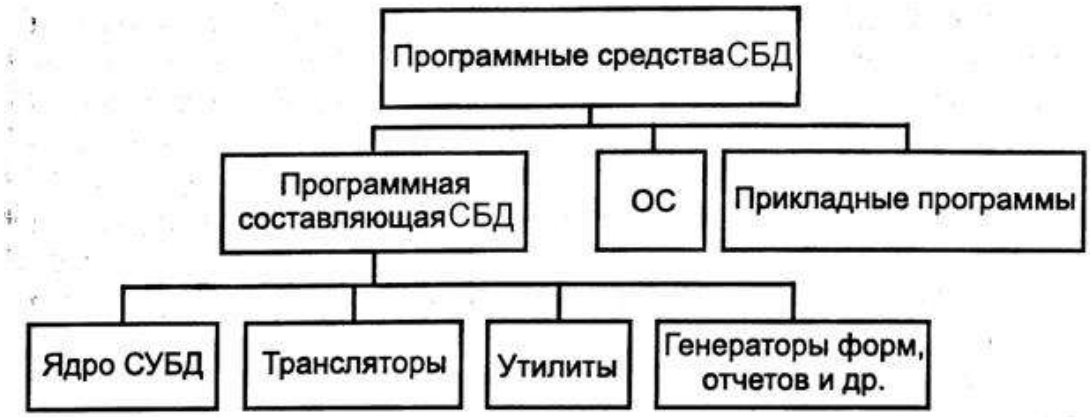

Рис. 1.4. Программные средства СБД

Программная составляющая СБД осуществляет обработку данных и взаимодействие с операционной системой (ОС) и прикладными программами. Взаимосвязь компонентов этого комплекса программных средств показана на рис. 1.5.

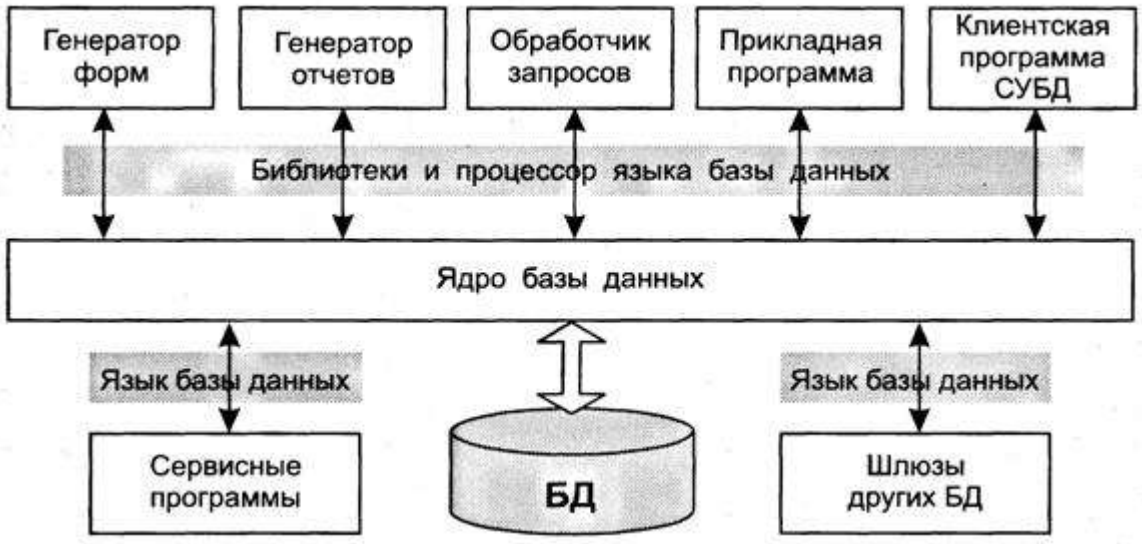

Рис. 1.5. Программная составляющая СБД

В составе комплекса выделяют следующие компоненты:

· *ядро,* обеспечивающее управление данными во внешней и оперативной памяти, а также протоколирование изменений;

· *процессор языка БД,* обеспечивающий обработку и оптимизацию запросов на выборку и изменение данных;

· *подсистему (библиотеку) поддержки программных вызовов,* которая обслуживает прикладные программы управления данными, взаимодействующие с СУБД через средства пользовательского интерфейса;

· *сервисные программы* (системные и внешние утилиты), обеспечивающие настройку СУБД, восстановление после сбоев и другое обслуживание.

Необходима координация между ОС и СУБД. Многопользовательские приложения, обработка распределенных запросов требуют эффективного использования ресурсов, которыми управляет ОС. Управление доступом и обеспечение защиты также интегрируются с соответствующими средствами ОС. К *техническим средствам СБД* относятся: ЭВМ, периферийные средства ввода информации в БД, средства хранения данных и отображения выводимой информации. Для сетевых СБД необходимы коммуникационные средства (рис. 1.6).

Рис. 1.6. Технические средства СБД

В качестве ЭВМ используются *универсальные компьютеры* и специализированные *серверы* – машины с повышенной отказоустойчивостью, высокопроизводительными подсистемами ввода-вывода и развитой периферией. Для распределенных БД важны коммуникационное оборудование и сетевые протоколы. Специализированные технические средства – машины баз данных и сетевые компьютеры без дисковых накопителей широкого

распространения не нашли. Карманные ПК используются в качестве коммуникационных устройств для доступа мобильных пользователей к корпоративным данным в глобальных сетях.

В СБД выполняются операции по вводу, хранению, обработке и выводу информации (рис. 1.7). При выполнении этих операций используются различные технологии и различные технические и программные средства.

Рис. 1.7. Операции с БД

Организационно-методические средства СБД представляют собой инструкции, методические и регламентирующие материалы для пользователей разных категорий. К ним же относятся методики проектирования БД.

1.3. Администрация базы данных

Создание, функционирование и развитие СБД обеспечиваются *администрацией базы данных (АБД)*, которая выполняет работы на протяжении всего жизненного цикла системы. В состав АБД входят:

· системные аналитики;

· проектировщики структур данных и внешнего по отношению к СБД информационного обеспечения;

· проектировщики технологических процессов обработки данных;

· системные и прикладные программисты;

· операторы;

· специалисты по техническому обслуживанию;

· специалисты по маркетингу (для коммерческих СБД).

В обязанности АБД входит выполнение следующих функций.

1. Анализ предметной области, ее описание, формулировка ограничений целостности, определение потребностей и статуса пользователей.

2. Проектирование структуры БД: определение состава и структуры информационных единиц БД, связей между ними.

3. Задание ограничений целостности при описании структуры БД и процедур обработки данных.

4. Первоначальная загрузка и ведение БД: разработка технологии загрузки и ведения БД, проектирование форм ввода, создание программных модулей.

5. Защита данных:

· обеспечение парольного входа в систему;

· определение прав доступа пользователей к данным;

· выбор и создание программно-технических средств защиты данных;

· тестирование средств защиты данных;

· сбор статистики об использовании данных;

· исследование случаев нарушения защиты данных;

· обеспечение восстановления БД, ведение системных журналов.

6. Анализ обращений пользователей к БД.

7. Работа с конечными пользователями.

8. Работа над совершенствованием и динамическим развитием СБД.

На рис. 1.8 представлена схема взаимодействия компонентов СБД в процессе создания и эксплуатации. Создание БД начинается с проектирования и описания на ЯОД (1). На этом этапе могут использоваться методики ручного проектирования и CASE-средства, автоматически генерирующие описания БД. Описания вводятся в СБД и запоминаются в соответствии с требованиями

конкретной СУБД (2,3). После того как описание БД сохранено, в нее вводятся данные (4). При этом СУБД использует метаинформацию, зафиксированную в словаре данных.

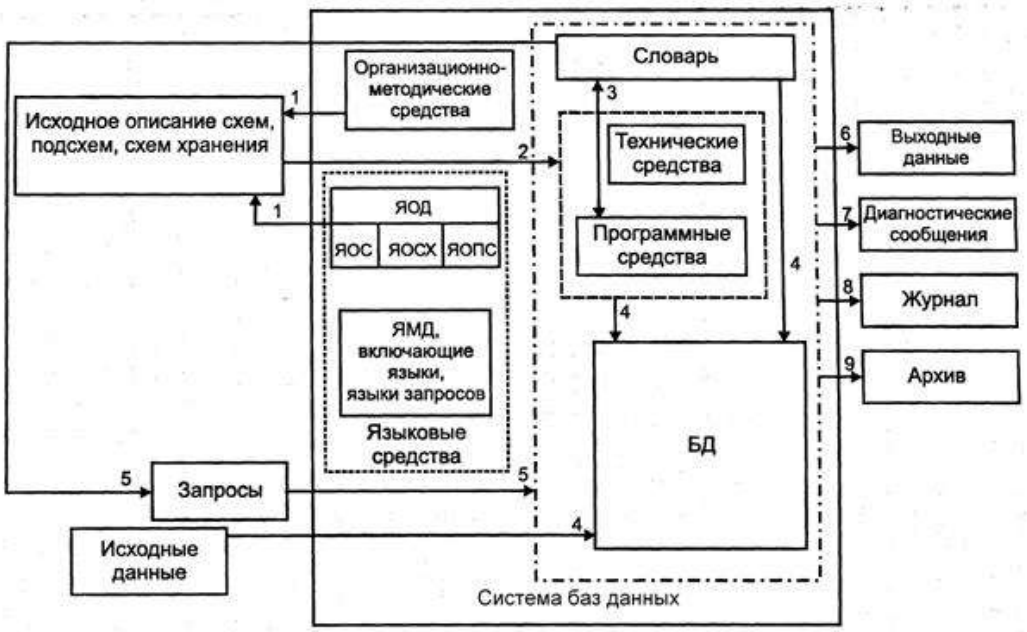

Рис. 1.8. Схема взаимодействия компонентов СБД

Заполненная БД используется для извлечения необходимой пользователям информации (5). При формулировании запросов используется информация, содержащаяся в схемах и подсхемах. В результате выполнения запроса пользователю выдаются выходные данные (6). Кроме данных выдается диагностическая информация (7). Для обеспечения надежности необходимо осуществлять журнализацию выполняемых действий (8) и регулярно архивировать данные (9).

1.4. Классификация баз данных

СБД являются сложными системами, и их классификация может быть произведена как для всей СБД, так и для каждого компонента отдельно (рис. 9). Центральным компонентом СБД является БД и большинство классификационных признаков относится к ней. *По форме представления информации* различают визуальные, а также системы аудио и мультимедиа. Эта классификация показывает, в каком виде информация хранится в БД и выдается пользователям.

По характеру организации данных БД могут быть разделены на неструктурированные, частично структурированные и структурированные. К неструктурированным могут быть отнесены БД, организованные в виде семантических сетей. Частично структурированными можно считать БД в виде обычного текста или гипертекстовые системы. Структурированные БД требуют предварительного проектирования и описания структуры. Структурированные БД *по типу используемой модели* делятся на иерархические, сетевые, реляционные, смешанные и мультимодельные. Эта классификация распространяется и на СУБД.

В структурированных БД различают несколько уровней *информационных единиц* (ИЕ), входящих одна в другую. Большинство систем поддерживает:

- *поле* – наименьшая семантическая единица информации;
- совокупность полей (или более сложных ИЕ) образует *запись*;
- множество однотипных записей представляет *файл базы данных*.

Многие СУБД в явном виде поддерживают и уровень базы данных как совокупности взаимосвязанных файлов БД. *По типу хранимой информации* БД делятся на фактографические, документальные и лексикографические. В *фактографических БД* хранится информация фактического характера – числовые или текстовые характеристики объектов, представленные в формализованном виде. В ответ на запрос выдается информация об интересующем объекте.

В *документальных БД* единицей хранения является документ и пользователю выдается ссылка на документ или сам документ. Документальные БД организуются без хранения и с хранением документа на машинных носителях. К первому типу относятся *библиографические*, *реферативные* и *БД-указатели*, отсылающие к источнику информации. Системы, хранящие полный текст документа, называются *полнотекстовыми*. Их разновидностью являются *БД форм документов*, в которых документ ищется для использования его в качестве шаблона.

К *лексикографическим БД* относятся различные словари

(классификаторы, многоязычные словари, словари основ слов и т. п.).

По характеру организации хранения данных и обращения к ним различают локальные (персональные), общие (интегрированные, централизованные) и распределенные БД (рис. 1.10).

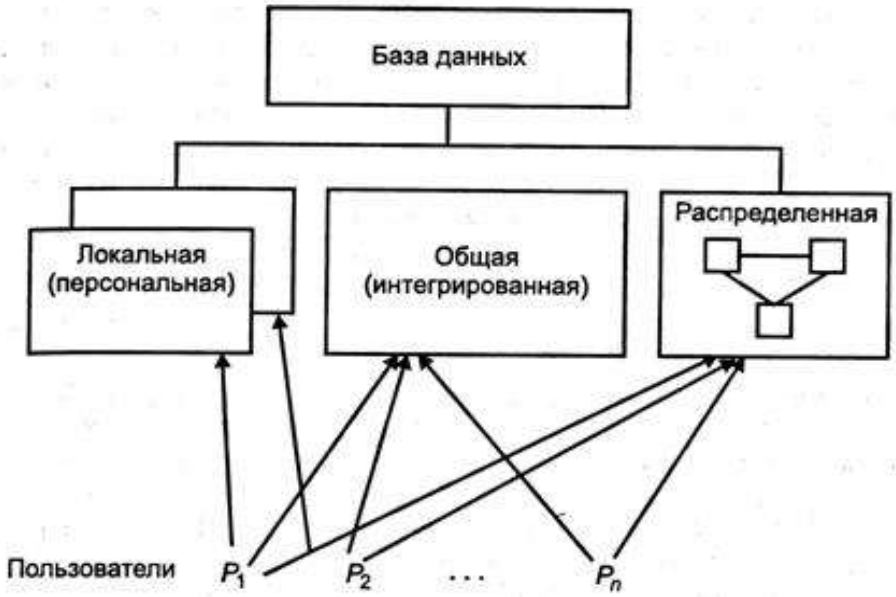

Рис. 1.10. Классификация БД по характеру хранения и обращения к данным

Персональная БД предназначена для локального использования одним пользователем. Локальные БД могут создаваться каждым пользователем самостоятельно, а могут извлекаться из общей БД.

Интегрированные и распределенные БД предполагают возможность одновременного обращения к информации нескольких пользователей (многопользовательский режим доступа). Части распределенных БД физически расположены на разных ЭВМ, но логически представляют собой единое целое.

Распределяться по узлам сети могут и другие компоненты СБД. Сама БД при этом может быть нераспределенной. Поэтому различают:

- распределенные БД;
- распределенные СБД (в которых распределен хотя бы один компонент).

В некоторых источниках упоминают *экстенсиональные* и *интенсиональные* БД. Первые строятся с помощью явного хранения данных в БД, вторые – с помощью правил, определяющих их содержание.

1.5. Классификация СУБД

По языкам общения СУБД делятся на открытые, замкнутые и смешанные. В открытых системах для обращения к БД используются универсальные языки. Замкнутые системы имеют собственные языки общения с пользователями СБД.

По выполняемым функциям СУБД делятся на информационные и операционные. Информационные позволяют организовать хранение информации и доступ к ней. Для более сложной обработки необходимы специальные программы. Операционные выполняют сложную обработку и могут менять алгоритмы обработки.

По сфере возможного применения различают универсальные и специализированные (проблемно ориентированные СУБД).

Набор типов данных в разных СУБД различен. Ряд СУБД позволяет разработчику добавлять новые типы данных и новые операции. Такие системы называются *расширяемыми системами баз данных*. Дальнейшим развитием являются *системы объектно-ориентированных баз данных,* обладающие мощными возможностями моделирования сложных объектов.

По ориентации на преобладающую категорию пользователей можно выделить СУБД для разработчиков и для конечных пользователей. Первые должны иметь качественные компиляторы и позволять создавать отчуждаемые программные продукты, обладать развитыми средствами отладки, включать средства документирования. Вторые должны иметь удобный интерфейс, высокий уровень языковых средств, интеллектуальные модули подсказок, защиту от ошибок и т. п.

Существует разделение СУБД *по поколениям.* К 1-му поколению относят системы, основанные на иерархической и сетевой моделях (1960–70-е гг.), ко 2-му поколению – реляционные системы. СУБД 3-го поколения должны поддерживать сложные структуры данных и более развитые средства обеспечения целостности данных, отвечать требованиям, предъявляемым к открытым системам.

По мощности СУБД делятся на настольные (Dbase, FoxBase/FoxPro,

Clipper, Paradox, Access, Approach) и корпоративные (Oracle, DB2, Sybase, Informix, Ingres, Progress). Для первых характерны невысокие требования к техническим средствам, ориентация на конечного пользователя и низкая стоимость. Вторые обеспечивают работу в распределенной среде, высокую производительность, имеют развитые средства администрирования, широкие возможности поддержания целостности. Они сложны, дороги и требуют значительных ресурсов. Среди СУБД, занимающих промежуточное положение между настольными и промышленными системами, можно назвать Interbase, Microsoft SQL Server. В последние годы наметилась тенденция к стиранию границ между настольными и профессиональными системами.

По степени распределённости

- Локальные СУБД (все части локальной СУБД размещаются на одном компьютере)
- Распределённые СУБД (части СУБД могут размещаться на двух и более компьютерах).

По способу доступа к БД

- Файл-серверные

В файл-серверных СУБД файлы данных располагаются централизованно на файл-сервере. СУБД располагается на каждом клиентском компьютере (рабочей станции). Доступ СУБД к данным осуществляется через локальную сеть. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок. Преимуществом этой архитектуры является низкая нагрузка на ЦП сервера. Недостатки: потенциально высокая загрузка локальной сети; затруднённость централизованного управления; затруднённость обеспечения таких важных характеристик как высокая надёжность, высокая доступность и высокая безопасность. Применяются чаще всего в локальных приложениях, которые используют функции управления БД.

На данный момент файл-серверные СУБД считаются устаревшими.

Примеры: Microsoft Access, Paradox, dBase, FoxPro, Visual FoxPro.

- Клиент-серверные

Клиент-серверная СУБД располагается на сервере вместе с БД и осуществляет доступ к БД непосредственно, в монопольном режиме. Все клиентские запросы на обработку данных обрабатываются клиент-серверной СУБД централизованно. Недостаток клиент-серверных СУБД состоит в повышенных требованиях к серверу. Достоинства: потенциально более низкая загрузка локальной сети; удобство централизованного управления; удобство обеспечения таких важных характеристик как высокая надёжность, высокая доступность и высокая безопасность.

Примеры: Oracle, Firebird, Interbase, IBM DB2, MS SQL Server, Sybase, PostgreSQL, MySQL, Caché.

Встраиваемая СУБД (англ. *embedded DBMS*) — СУБД, которая может поставляться как составная часть некоторого программного продукта, не требуя процедуры самостоятельной установки. Встраиваемая СУБД предназначена для локального хранения данных своего приложения и не рассчитана на коллективное использование в сети. Физически встраиваемая СУБД чаще всего реализована в виде подключаемой библиотеки. Доступ к данным со стороны приложения может происходить через SQL либо через специальные программные интерфейсы.

Примеры: OpenEdge, SQLite, BerkeleyDB, Firebird Embedded, MySQL, Sav Zigzag, Microsoft SQL Server Compact, ЛИНТЕР.

Тема 2. Реляционная модель данных

Основные вопросы:
2.1. Модели данных
2.2. Основные понятия реляционной модели данных
2.3. Целостность реляционных данных
2.4. Операции над отношениями
2.5. Нормализация баз данных
2.6. Распределенные базы данных

2.1. Модели данных

Ядром БД является *модель данных* – совокупность структур данных и операций их обработки. Различают иерархическую, сетевую и реляционную модели.

Иерархическая модель позволяет строить БД с древовидной структурой. В них каждый узел содержит свой тип данных (сущность). На верхнем уровне дерева имеется один узел – корень, на следующем уровне располагаются узлы, связанные с этим корнем, затем узлы, связанные с узлами предыдущего уровня, и т. д. Каждый узел может иметь только одного предка (рис. 2.1).

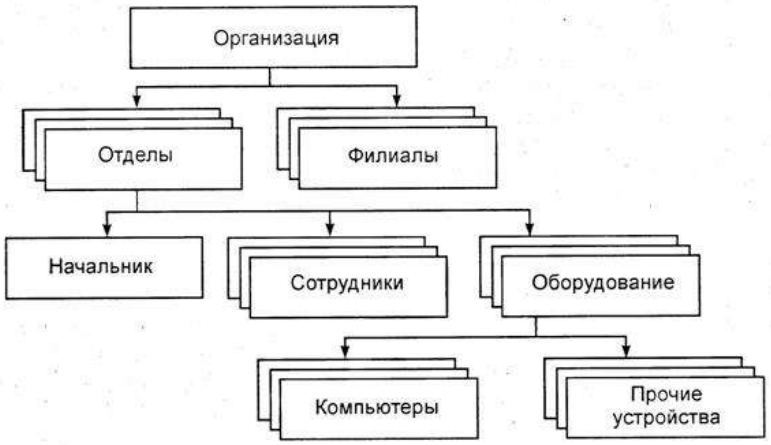

Рис. 2.1 Иерархическая древовидная структура модели БД

Поиск данных всегда начинается с корня. Затем производится спуск с одного уровня на другой пока не будет достигнут искомый уровень. Перемещения от одной записи к другой осуществляются с помощью ссылок. Достоинствами иерархической модели являются простота описания иерархических структур реального мира, а также быстрое выполнение запросов, соответствующих структуре данных. Недостатки иерархической

модели в том, что они часто содержат избыточные данные и не всегда удобно каждый раз начинать поиск нужных данных с корня.

В *сетевой модели* возможны связи всех информационных объектов со всеми. Например, каждый преподаватель может обучать много студентов и каждый студент может обучаться у многих преподавателей (рис. 2.2).

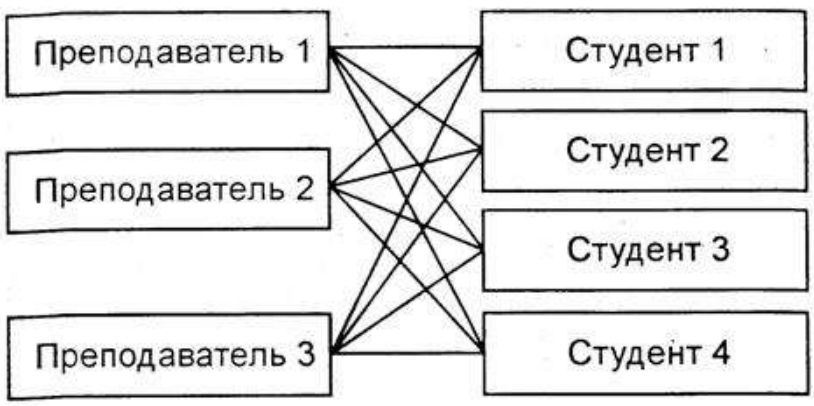

Рис. 2.2. Сетевая структура модели БД

Использование иерархической и сетевой моделей ускоряет доступ к информации, но требует значительных ресурсов памяти, так как каждый элемент данных содержит ссылки на другие элементы. Характерна сложность реализации СУБД.

Реляционная модель (РМД) была разработана в начале 1970-х годов Эдгаром Ф. Коддом. В ней информация представляется в виде двумерных таблиц, а операции сводятся к манипуляциям с таблицами. В 1980-х годах она получила широкое распространение, а реляционные СУБД стали промышленным стандартом. Причины доминирования РМД обусловлены тем, что имеются

- развитая теория (реляционная алгебра);
- · аппарат сведения других моделей данных к РМД;
- ·специальные средства ускоренного доступа к информации;
- · стандартизированный высокоуровневый язык запросов к БД, позволяющий манипулировать данными без знания физической организации БД.

Объектно-ориентированная модель начала разрабатываться в 1990-е годы с появлением объектно-ориентированных языков. Такие БД хранят

методы классов, что позволяет интегрировать данные и их обработку в приложениях.

2.2. Основные понятия реляционной модели данных

В математических дисциплинах понятию «таблица» соответствует понятие «отношение» (relation). Таблица отражает объект реального мира – сущность, а каждая ее строка отражает конкретный экземпляр сущности. Каждый столбец имеет уникальное для таблицы имя. Строки не имеют имен, порядок их следования не определен, а количество логически не ограничено. Одним из основных преимуществ РМД является однородность (каждая строка таблицы имеет один формат). Пользователь сам решает вопрос, обладают ли соответствующие сущности однородностью. Этим решается проблема пригодности модели. Основные элементы РМД показаны на рис. 2.3.

Отношение представляет собой двумерную таблицу, содержащую некоторые данные. Сущность – объект любой природы, данные о котором хранятся в БД. Атрибуты – свойства, характеризующие сущность (столбцы). Степень отношения – количество столбцов. Схема отношения – список имен атрибутов, например, *СОТРУДНИК (№, ФИО, Год рождения, Должность, Кафедра)*. Домен – совокупность значений атрибутов отношения (тип данных). Кортеж – строка таблицы. Кардинальность (мощность) – количество строк в таблице.

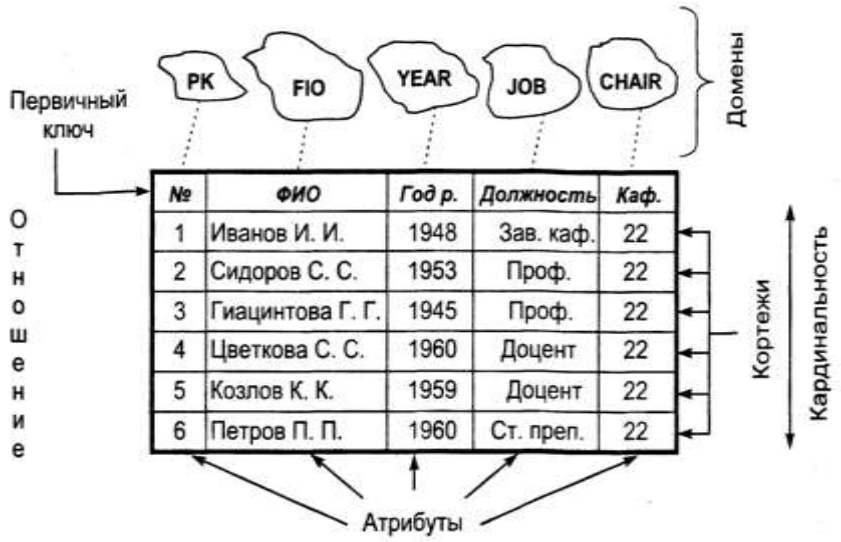

Рис 2.3. Элементы реляционной модели

Первичный ключ – это атрибут, уникально идентифицирующий строки отношения. Первичный ключ из нескольких атрибутов называется составным. Первичный ключ не может быть полностью или частично пустым (иметь значение null). Ключи, которые можно использовать в качестве первичных, называются *потенциальными* или *альтернативными ключами*. *Внешний ключ* – это атрибут (атрибуты) одной таблицы, который может служить первичным ключом другой таблицы. Является ссылкой на первичный ключ другой таблицы (рис. 2.4).

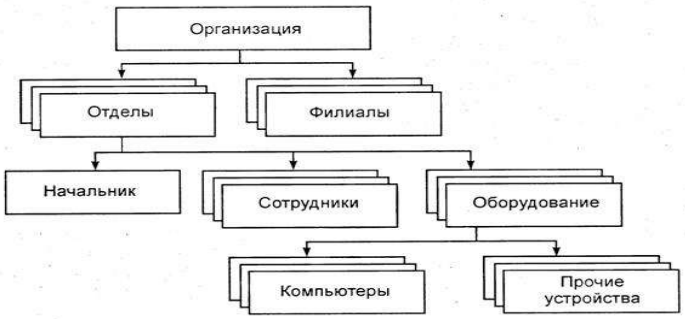

Рис. 2.4. Иерархическая древовидная структура модели БД

Поиск данных всегда начинается с корня. Затем производится спуск с одного уровня на другой пока не будет достигнут искомый уровень. Перемещения от одной записи к другой осуществляются с помощью ссылок.

Достоинствами иерархической модели являются простота описания иерархических структур реального мира, а также быстрое выполнение запросов, соответствующих структуре данных. Недостатки иерархической модели в том, что они часто содержат избыточные данные и не всегда удобно каждый раз начинать поиск нужных данных с корня.

В *сетевой модели* возможны связи всех информационных объектов со всеми. Например, каждый преподаватель может обучать много студентов и каждый студент может обучаться у многих преподавателей (рис. 2.5).

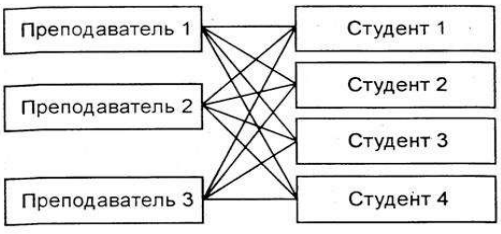

Рис. 2.5. Сетевая структура модели БД

Использование иерархической и сетевой моделей ускоряет доступ к информации, но требует значительных ресурсов памяти, так как каждый элемент данных содержит ссылки на другие элементы. Характерна сложность реализации СУБД.

Реляционная модель (РМД) была разработана в начале 1970-х годов Эдгаром Ф. Коддом. В ней информация представляется в виде двумерных таблиц, а операции сводятся к манипуляциям с таблицами. В 1980-х годах она получила широкое распространение, а реляционные СУБД стали промышленным стандартом. Причины доминирования РМД обусловлены тем, что имеются:

Рис 2.6. Связь отношений

Отношения *СТУДЕНТ (ФИО, Группа, Специальность)* и *ПРЕДМЕТ (Назв Пр, Часы)* связаны отношением *СТУДЕНТ_ПРЕДМЕТ (ФИО, Назв Пр, Оценка)*, в котором внешние ключи *ФИО* и *Назв_Пр* образуют составной ключ.

2.3. Целостность реляционных данных

Логические ограничения, накладываемые на данные, называются *ограничениями целостности.* СУБД должна контролировать соответствие данных заданным ограничениям при переводе БД из одного состояния в другое. Использование ограничений связано также с адекватностью отражения предметной области. *Явные ограничения* задаются семантикой предметной области. Они описывают области допустимых значений атрибутов, соотношение между атрибутами, динамику их изменения и т. д. *Внутренние ограничения* свойственны собственно модели данных. Они накладываются на структуру отношений, на связи, на допустимые значения наборов данных, заложенные в выбранной модели данных. Способы реализации внутренних ограничений целостности зависят от СУБД.

В РМД существует два вида внутренних ограничений целостности.

1. Целостность по существованию – потенциальный ключ отношения не может иметь пустого значения (NULL).

2. Целостность по связи – определяется понятием внешнего ключа отношения и означает систему правил, используемых для поддержания связей между записями в связанных таблицах. Обеспечивает защиту от случайного удаления или изменения связанных данных, от некорректного изменения ключевых полей.

2.4. Операции над отношениями

РМД стала первой работоспособной моделью данных, поскольку имела эффективный инструментарий – операции реляционной алгебры. Основной единицей обработки является отношение, а не его кортежи. К отношениям можно применить *систему операций*, позволяющих получить одни отношения из других. Исключение составляют операции создания и заполнения таблиц, а также операции описания и переименования столбцов. Результатом запроса к реляционной БД может быть новое отношение, вычисленное на основе имеющихся отношений.

Реляционная алгебра включает две группы операций.

1. Традиционные операции над множествами (модифицированные с учетом того, что их операндами являются отношения) – объединение, пересечение, разность (вычитание), декартово произведение и деление.

2. Специальные реляционные операции – выборка, проекция, соединение.

Объединение выполняется над двумя совместными отношениями *R1, R2* с идентичной структурой. В результате операции строится новое отношение *R = R1 U R2* , которое имеет тот же состав атрибутов и совокупность кортежей исходных отношений. В результирующее отношение по определению не включаются дубликаты кортежей. Ниже приведены исходные отношения: *R1* (табл. 2.1) и *R2* (табл. 2.2) и результат объединения – *R* (табл. 2.3).

Таблица 2.1

ФИО	Год рождения	Должность	Кафедра
Иванов И.И.	1948	Зав. кафедрой	22
Сидоров С.С.	1953	Доцент	22
Козлов К.К.	1980	Ассистент	23

Таблица 2.2

ФИО	Год рождения	Должность	Кафедра
Цветкова Н.Н.	1965	Доцент	23
Петрова П.П.	1953	Ст. преподаватель	22
Козлов К.К.	1980	Ассистент	23

Таблица 2.3

ФИО	Год рождения	Должность	Кафедра
Иванов И.И.	1948	Зав. кафедрой	22
Сидоров С.С.	1953	Доцент	22
Козлов К.К.	1980	Ассистент	23
Цветкова Н.Н.	1965	Доцент	23
Петрова П.П.	1953	Ст. преподаватель	22

Пересечение выполняется над двумя совместными отношениями *R1, R2.* Результирующее отношение *RP = R1 R2* содержит кортежи, которые есть в каждом из исходных. Результат имеет тот же состав атрибутов, что и исходные отношения. Пересечение отношений *R1* и *R2* дает отношение *RP* (табл. 2.4).

Таблица 2.4

ФИО	Год рождения	Должность	Кафедра
Козлов К.К.	1980	Ассистент	23

Вычитание выполняется над двумя совместными отношениями *R1, R2.* В результате строится новое отношение *RV = R1 - R2* с идентичным набором атрибутов, содержащее кортежи первого отношения *R1*, которые не входят в отношение *R2.* Вычитание отношения *R2* из *R1* дает отношение *RV* (табл. 2.5).

Таблица 2.5

ФИО	Год рождения	Должность	Кафедра
Иванов И.И.	1948	Зав. кафедрой	22
Сидоров С.С.	1953	Доцент	22

Декартово произведение выполняется над двумя отношениями *R1* и *R2,* имеющими в общем случае разный состав атрибутов. В результате образуется новое отношение *RD = R1 x R2,* которое включает все атрибуты исходных

отношений. Результирующее отношение состоит из всевозможных сочетаний кортежей исходных отношений. Число кортежей (мощность) отношения-произведения равно произведению мощностей исходных отношений.

Декартово произведение отношений *R1* (табл. 2.6) и *R2* (табл. 2.7) дает новое отношение *RD* (табл. 2.8), которое содержит все атрибуты исходных отношений. В него целесообразно добавить атрибут *Оценка* для записи результатов экзамена.

Таблица 2.6

Номер студента	ФИО студента
11	Иванов И.И.
12	Петров П.П.
13	Сидоров С.С.

Таблица 2.7

Код дисциплины	Наименование
Д1	Математика
Д2	Информатика

Таблица 2.8

Номер студента	ФИО студента	Код дисциплины	Наименование	Оценка
11	Иванов И.И.	Д1	Математика	5
12	Петров П.П.	Д1	Математика	3
13	Сидоров С.С.	Д1	Математика	5
11	Иванов И.И.	Д2	Информатика	5
12	Петров П.П.	Д2	Информатика	4
13	Сидоров С.С.	Д2	Информатика	4

Деление выполняется над двумя отношениями *R1* и *R2,* имеющими в общем случае разные структуры и часть одинаковых атрибутов. В результате образуется новое отношение, содержащее атрибуты 1-го операнда, отсутствующие во 2-м операнде, и кортежи 1-го операнда, которые совпали с кортежами 2-го. Для выполнения этой операции 2-й операнд должен содержать лишь атрибуты, совпадающие с атрибутами 1-го.

Например, чтобы узнать, кто из студентов получил по математике 5 и по информатике 4, надо разделить отношения *Экзаменационная ведомость* на вспомогательное отношение *Мат5Физ4 (Наименование, Оценка)* с двумя кортежами: *Математика, 5* и *Информатика, 4*. В результате получим отношение *Итог (Номер студента, ФИО студента, Код дисциплины)* с одним кортежем – *13, Сидоров, Д1*.

Выборка выполняется над одним отношением *R*. Для отношения по заданному условию (предикату) осуществляется выборка подмножества кортежей. Результирующее отношение имеет ту же структуру, что и исходное, но число его кортежей будет меньше (или равно) числа кортежей исходного отношения. Например, выбрать студентов, сдавших математику на отлично *(Код дисциплины = Д1) AND (Оценка = 5)*.

Операция соединения имеет большое значение для РБД, так как в процессе нормализации отношений исходное отношение разбивается на несколько более мелких отношений, которые при выполнении запросов пользователя требуется, как правило, вновь соединять для восстановления исходного отношения.

Рассмотренные выше операции в той или иной мере реализуются в языке манипулирования данными СУБД (SQL, QBE, другие языки запросов). Язык SQL является более чем реляционно-полным, так как кроме операций реляционной алгебры содержит полный набор операторов над кортежами – *Включить, Удалить, Изменить,* а также реализует арифметические операции и операции сравнения.

К достоинствам РМД относятся:

- простота представления данных благодаря табличной форме;
- минимальная избыточность данных при нормализации отношений;
- обеспечение независимости приложений пользователя от данных, допускающей включение или удаление отношений, изменение их атрибутного состава.

К недостаткам РМД можно отнести то, что нормализация данных приводит к значительной их фрагментации, в то время как в большинстве задач необходимо объединение фрагментированных данных.

2.5. Нормализация баз данных

Данные могут группироваться в таблицы (отношения) разными способами. При проектировании БД в качестве отправной точки может использоваться

одно *универсальное отношение*, в которое включаются все необходимые атрибуты. Оно может содержать все данные, которые предполагается размещать в БД. В качестве примера рассмотрим универсальное отношение *сотрудники*, содержащее информацию о сотрудниках предприятия (табл. 2.9).

Таблица 2.9

Код сотрудника	ФИО	Должность	Номер отдела	Наименование отдела	Квалификация
7513	Иванов И.И.	Программист	128	Отдел проектирования	C, Java
9842	Сергеева С.С.	Администратор БД	42	Финансовый отдел	DB2
6651	Петров П.П.	Программист	128	Отдел проектирования	VB, Java
9006	Николаев Н.Н.	Системный администратор	128	Отдел проектирования	Windows, Linux

При использовании универсального отношения возникают две проблемы:

- избыточность данных;
- потенциальная противоречивость (аномалии).

Под *избыточностью* понимают повторение данных в разных строках одной таблицы или в разных таблицах БД. Так, для каждого сотрудника отдела 128 повторяются данные «128, Отдел проектирования».

Аномалии – это проблемы, возникающие в данных из-за дефектов проектирования БД. Существуют три вида аномалий: вставки, удаления и модификации. *Аномалии вставки* проявляются при вводе данных в дефектную таблицу. Добавляя информацию о новом сотруднике, мы должны добавить номер и название отдела. Если ввести данные, не соответствующие имеющимся в таблице (например, 42, отдел проектирования), будет не ясно, какая из строк БД содержит правильную информацию. *Аномалии удаления* возникают при удалении данных из дефектной схемы. Предположим, что все сотрудники отдела 128 уволились в один и тот же день. После удаления записей этих сотрудников в БД больше не будет ни одной записи, содержащей информацию об отделе 128. *Аномалии модификации* возникают при изменении данных дефектной схемы. Предположим, что отдел 128 решили переименовать в отдел передовых технологий. Необходимо изменить соответствующие данные о

каждом сотруднике отдела. Если мы пропустим хотя бы одну запись, возникнет аномалия модификации.

Правилом разработки хорошей структуры БД является необходимость избегать схем с большим числом *пустых атрибутов*. Если мы хотим указать, что один из ста служащих имеет особую квалификацию, для хранения этой информации не следует добавлять в таблицу еще один столбец, поскольку для остальных 99 работников значением столбца будет NULL. Вместо этого следует добавить новую таблицу, в которой будут храниться только кодовые номера и информация о квалификации тех работников, которых это касается.

Решение перечисленных проблем состоит в разделении данных и связей, что обеспечивается процедурой *нормализации*. Концепции и методы нормализации были разработаны Э. Ф. Коддом.

Нормализация отношений – это формальный аппарат ограничений на формирование отношений, который позволяет устранить дублирование и потенциальную противоречивость хранимых данных, уменьшает трудозатраты на ведение БД. Процесс нормализации заключается в декомпозиции исходных отношений на более простые отношения. Цель нормализации – получение такого проекта БД, в котором «каждый факт появляется лишь в одном месте».

Теория нормализации основана на наличии зависимостей между атрибутами отношения. Основными видами зависимостей являются:

· функциональные;

· многозначные;

· транзитивные.

Базовым является понятие *функциональной зависимости*, поскольку на его основе формируются определения всех остальных видов зависимостей. Атрибут B функционально зависит от атрибута A, если каждому значению A соответствует в точности одно значение B. Математически функциональную зависимость B от A обозначают A ’ B. Это означает, что во всех кортежах с одинаковым значением атрибута A атрибут B будет иметь также одно и то же значение. При этом A и B могут быть составными, то есть состоять из двух и более атрибутов.

Зависимость, при которой каждый неключевой атрибут зависит от всего составного ключа и не зависит от его частей, называется *полной функциональной зависимостью.* Если атрибут А зависит от атрибута В, а атрибут В зависит от атрибута С (С ’ В ’ А), но обратная зависимость отсутствует, то зависимость А от С называется *транзитивной.*

Многозначная зависимость. Говорят, что один атрибут отношения многозначно определяет другой атрибут того же отношения, если для каждого значения первого атрибута существует множество соответствующих значений второго атрибута. Многозначные зависимости могут быть:

- один-ко-многим (1:М);
- многие-к-одному (М:1);
- многие-ко-многим (М:М).

Каждая ступень процесса нормализации приводит схему отношений в последовательные нормальные формы. Для каждой ступени имеются наборы ограничений. Выделяют следующую последовательность нормальных форм:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);
- третья нормальная форма (3НФ);
- усиленная 3НФ или нормальная форма Бойса-Кодда (БКНФ);
- четвертая нормальная форма (4НФ);
- пятая нормальная форма (5НФ).

Отношение находится *в первой нормальной форме (1НФ)*, когда каждая строка содержит только одно значение для каждого атрибута (столбца), то есть все атрибуты отношения имеют единственное значение (являются атомарными). В столбце *Квалификация* ненормализованной таблицы содержатся списки значений (С, Java и т. д.). Чтобы привести схему к 1НФ, необходимо разместить в этом столбце атомарные значения. Самый простой способ заключается в выделении по одной строке на каждый элемент квалификации (табл. 2.10).

Таблица 2.10

Код сотрудника	ФИО	Должность	Номер отдела	Наименование отдела	Квалификация
7513	Иванов И.И.	Программист	128	Отдел проектирования	C
7513	Иванов И.И.	Программист	128	Отдел проектирования	Java
9842	Сергеева С.С.	Администратор БД	42	Финансовый отдел	DB2
6651	Петров П.П.	Программист	128	Отдел проектирования	VB
6651	Петров П.П.	Программист	128	Отдел проектирования	Java
9006	Николаев Н.Н.	Системный администратор	128	Отдел проектирования	Windows
9006	Николаев Н.Н.	Системный администратор	128	Отдел проектирования	Linux

Такое решение далеко от идеального, поскольку порождает очевидную избыточность данных – для каждой комбинации сотрудник-квалификация приходится хранить все характеристики сотрудника. Отношение находится *во второй нормальной форме (2НФ)*, если оно находится в 1НФ, и каждый неключевой атрибут полностью функционально зависит от всех составляющих первичного ключа. Если атрибут не зависит полностью от первичного ключа, то он внесен ошибочно и должен быть удален. Нормализация производится путем нахождения существующего отношения, к которому относится данный атрибут, или созданием нового отношения, в который атрибут должен быть помещен.

Таблица *Квалификации_сотрудников* (табл. 2.10) находится в 1НФ, но не удовлетворяет 2НФ. Первичный ключ должен уникальным образом идентифицировать каждую строку. Единственным вариантом является использование в качестве первичного ключа комбинации *Код сотрудника* и *Квалификация*. Это порождает схему: *Квалификации_сотрудников (Код сотрудника, ФИО, Должность, Номер отдела, Наименование отдела, Квалификация)*.

Одной из имеющихся здесь функциональных зависимостей будет: *Код*

сотрудника, Квалификация ’ ФИО, Должность, Номер отдела, Наименование отдела. Но, кроме того, мы также имеем: *Код сотрудника ’ ФИО, Должность, Номер отдела, Наименование отдела.* Другими словами, можно определить имя, должность и отдел, используя только код сотрудника. Это значит, что указанные атрибуты функционально зависимы только от части первичного ключа, а не от всего первичного ключа. Следовательно, указанная схема не находится в 2НФ.

Для приведения этой схемы в 2НФ необходимо декомпозировать исходное отношение на два, в которых все неключевые атрибуты будут полностью функционально зависеть от ключа: *сотрудники (Код сотрудника, ФИО, Должность, Номер отдела, Наименование отдела)* и *Квалификации_сотрудников (Код сотрудника, Квалификация)* (табл. 2.11 и 2.12).

Таблица 2.11

Код сотрудника	ФИО	Должность	Номер отдела	Наименование отдела
7513	Иванов И.И.	Программист	128	Отдел проектирования
9842	Сергеева С.С.	Администратор БД	42	Финансовый отдел
6651	Петров П.П.	Программист	128	Отдел проектирования
9006	Николаев Н.Н.	Системный администратор	128	Отдел проектирования

Таблица 2.12

Код сотрудника	Квалификация
7513	C
7513	Java
9842	DB2
6651	VB
6651	Java
9006	Windows
9006	Linux

Отношение находится *в третьей нормальной форме (3НФ)*, если оно находится во 2НФ и ни один из его неключевых атрибутов не связан функциональной зависимостью с любым другим неключевым атрибутом. Атрибуты, зависящие от других неключевых атрибутов, нормализуются путем перемещения зависимого атрибута и атрибута, от которого он зависит, в новое отношение.

Формально, для приведения схемы в 3НФ необходимо исключить все

транзитивные зависимости. Схема отношения *сотрудники* (табл. 2.14) содержит следующие функциональные зависимости: *Код сотрудника ’ ФИО, Должность, Номер отдела, Наименование отдела* и *Номер отдела ’ Наименование отдела*.

Первичным ключом является *Код сотрудника*, и все атрибуты полностью функционально зависимы от него (первичный ключ определяется единственным атрибутом). При этом *Номер отдела* ключом не является.

Функциональная зависимость *Код сотрудника ’ Наименование отдела* является транзитивной, поскольку содержит промежуточный шаг (зависимость *Номер отдела ’ Наименование отдела*). Для приведения в 3НФ необходимо исключить эту транзитивную зависимость, декомпозируя отношение на два: *сотрудники (Код сотрудника, ФИО, Должность, Номер отдела)* и *отделы (Номер отдела, Наименование отдела)* (табл. 2.13).

Таблица 2.13

Код сотрудника	ФИО	Должность	Номер отдела
7513	Иванов И.И.	Программист	128
9842	Сергеева С.С.	Администратор БД	42
6651	Петров П.П.	Программист	128
9006	Николаев Н.Н.	Системный администратор	128

Номер отдела	Наименование отдела
42	Финансовый отдел
128	Отдел проектирования

Нормальная форма Бойса-Кодда (БКНФ) является развитием 3НФ и требует, чтобы в отношении были только такие функциональные зависимости, левая часть которых является потенциальным ключом отношения. *Потенциальный ключ* представляет собой атрибут (или множество атрибутов), который может быть использован для данного отношения в качестве первичного ключа. Фактически первичный ключ – это один из потенциальных ключей, назначенный в качестве первичного. *Детерминантом* называется левая часть функциональной зависимости. Отношение находится в БКНФ тогда и

только тогда, когда каждый детерминант отношения является потенциальным ключом. Алгоритм приведения ненормализованных схем в 3НФ показан на рис. 2.7. На практике построение 3НФ в большинстве случаев является достаточным и приведением к ней процесс построения реляционной БД заканчивается. Запомнить правила нормализации помогает изречение: «Нормализация – это ключ, целый ключ и ничего, кроме ключа».

Нормальные формы высших порядков (4НФ и 5НФ) представляют больший интерес для теоретических исследований, чем для практики проектирования БД. В них учитываются многозначные зависимости между атрибутами. *Полной декомпозицией отношения* называют такую совокупность произвольного числа его проекций, соединение которых позволяет получить исходное отношение.

Рис. 2.7. Алгоритм приведения ненормализованных схем в 3НФ

Отношение находится *в пятой нормальной форме (5НФ)*, когда в каждой его полной декомпозиции все проекции содержат возможный ключ. Отношение, не имеющее ни одной полной декомпозиции, также находится в 5НФ.

Четвертая нормальная форма (4НФ) является частным случаем 5НФ, когда полная декомпозиция должна быть соединением ровно двух проекций. На практике непросто подобрать отношение, которое находится в 4НФ, не будучи в 5НФ. Нормализация – это процесс последовательной замены отношения его

полными декомпозициями до тех пор, пока все они не будут находиться в 5НФ. Приведя отношения к БКНФ, можно с большой гарантией считать, что они находятся в 5НФ. Единственными функциональными зависимостями в любом отношении должны быть зависимости вида К ’ А, где К – первичный ключ, а А – атрибут.

2.6. Распределенные базы данных

Распределенная база данных (DDB – Distributed DataBase) – это совокупность множества взаимосвязанных БД, распределенных в компьютерной сети. БД распределена физически, но логически едина (имеет общую схему данных). В системах с распределенными БД используются разные технологии распределения данных по узлам сети – фрагментация и тиражирование. При использовании *фрагментации* единая логическая БД разбивается на составные части (фрагменты), хранящиеся в разных узлах сети. Разбиение может проводиться по территориальному, функциональному и временному критериям. При использовании *тиражирования* в нескольких узлах сети создаются и поддерживаются в согласованном состоянии (синхронизируются) копии всей БД или ее фрагментов. Копия БД называется *репликой.* В системах с централизованной БД (много клиентов/один сервер) проблемы управления БД решаются просто, поскольку вся она хранится на сервере. В системах с распределенной БД проектирование, реализация запросов и управление представляют собой сложные задачи. Однако такие системы обеспечивают большую гибкость, надежность и быстродействие. Технологии распределения данных видоизменяют преимущества и недостатки систем. Одно из преимуществ БД – сокращение дублирования – теряется при использовании тиражирования. При этом сохраняются возможности контроля целостности данных для всей системы.

Ведущими поставщиками СУБД сформулированы следующие свойства идеальной системы управления распределенными БД:

- *прозрачность относительно расположения данных* – СУБД должна

представлять все данные так, как если бы они были локальными;

· *гетерогенность системы* – СУБД должна работать с данными, которые хранятся в системах с различной архитектурой и производительностью;

· *прозрачность относительно сети* – СУБД должна одинаково работать в условиях разнородных сетей;

· *поддержка распределенных запросов* – пользователь должен иметь возможность объединять данные из любых баз, даже размещенных в разных системах;

· *поддержка распределенных изменений* – пользователь должен иметь возможность изменять данные в любых базах, на доступ к которым у него есть права;

· *поддержка распределенных транзакций* – СУБД должна выполнять транзакции, выходящие за рамки одной вычислительной системы, и поддерживать целостность БД при возникновении отказов как в системах, так и в сети;

· *безопасность* – СУБД должна обеспечивать защиту всей распределенной БД от несанкционированного доступа;

· *универсальность доступа* – СУБД должна обеспечивать единую методику доступа ко всем данным.

и одна из существующих СУБД не достигает этого идеала поскольку:

· низкая и несбалансированная производительность сетей снижает общую производительность обработки в распределенных транзакциях;

· обеспечение целостности данных в распределенных транзакциях базируется на принципе «все или ничего» и требует специального протокола, что приводит к длительной блокировке изменяемых данных;

· необходимо обеспечить совместимость данных, для хранения которых в разных системах используются разные форматы и кодировки;

· если каталог хранится в одной системе, то удаленный доступ будет замедлен; если будет размножен – изменения придется синхронизировать;

· необходимо обеспечить совместимость СУБД разных типов и

поставщиков;

· велика потребность в ресурсах для обнаружения и устранения тупиковых ситуаций в распределенных транзакциях.

Эти причины определили поэтапность введения в СУБД возможностей распределенной обработки. В простейшем случае пользователь может обращаться по сети к записям в БД на других компьютерах. В других случаях СУБД сама проводит аутентификацию удаленного клиента и устанавливает сетевое соединение.

Изучая тенденции развития технологий обработки данных, можно выделить два класса систем:

· системы распределенной обработки данных;

· системы распределенных баз данных.

Системы распределенной обработки данных базировались на многопользовательских операционных системах с БД на центральном компьютере. Клиентские места реализовались в виде терминалов, не имеющих собственных ресурсов. Развитие сетевых технологий, распространение персональных ЭВМ и стандартов открытых систем привело к появлению *систем распределенных БД*, размещенных в сети разнотипных компьютеров. Система состоит из узлов, каждый из которых является СУБД. БД любого узла доступна пользователю.

При обработке данных в сетевой среде выделяют следующие группы функций:

· презентационная логика (PL – Presentation Logic): ввод и отображение данных – внешний (пользовательский) уровень реализации функциональной обработки и представления;

· бизнес-логика (BL – Business Logic): функциональная обработка, реализующая алгоритм решения задач пользователя;

· манипулирование данными БД в рамках приложения, которое обычно реализуется средствами SQL (DL – Database Logic);

· управление данными и другими ресурсами БД, реализуемое внутренними

средствами конкретной СУБД обычно в рамках файловой системы ОС;

· управление процессами обработки: связывание и синхронизация процессов обработки данных разного уровня.

Взаимодействие пользователя с БД строится на основе модели «клиент – сервер». Клиентская часть отвечает за целевую обработку данных и организацию взаимодействия с пользователем. Серверная часть обеспечивает хранение данных, обрабатывает запросы и посылает результаты клиенту для специальной обработки. В общем случае эти части функционируют на отдельных компьютерах, т. е. к серверу БД с помощью сети подключены компьютеры клиентов.

Разделение процесса на клиентскую и серверную компоненты позволяет:

· одновременно использовать БД различным прикладным программам;

· централизовать функции управления, такие как защита информации, обеспечение целостности данных, управление совместным использованием ресурсов;

· обеспечивать параллельную обработку запроса в распределенных БД;

· высвобождать ресурсы рабочих станций и сети;

· повышать эффективность управления данными за счет использования специальных серверов баз данных.

В *архитектуре «файл-сервер»* (рис. 2.8) средства организации и управления БД (в том числе СУБД) располагаются на машине клиента, а БД, представляющая собой набор специализированных файлов, – на машине-сервере.

Рис. 2.8. Архитектура «файл – сервер»

В этом случае серверная компонента представлена даже не средствами

СУБД, а сетевыми составляющими ОС, обеспечивающими удаленный разделяемый доступ к файлам. Запрос к БД, сформулированный на языке манипулирования данными, преобразуется СУБД в последовательность команд ввода-вывода, которые обрабатываются операционной системой машины-сервера.

Недостатки архитектуры:

· высокая загрузка сети и машин-клиентов, так как обмен идет на уровне единиц информации файловой системы (физических записей, блоков, файлов);

· низкий уровень защиты данных, так как доступ к файлам БД управляется общими средствами ОС сервера;

· бизнес-правила функциональной обработки, сосредоточенные в клиентской части, могут быть противоречивыми.

Для схемы характерно наибольшее суммарное время обработки информации.

В *архитектуре «выделенный сервер базы данных»* (рис. 2.9) средства управления базой данных и база данных размещены на машине-сервере (DB-сервер).

Рис. 2.9. Архитектура с выделенным сервером базы данных

Обращение к БД осуществляется на *языке SQL*, поэтому сервер БД часто называют *SQL-сервером*. Он поддерживается всеми реляционными СУБД (Oracle, Informix, MS SQL, DB2, ADABAS D, InterBase, SyBase). Сервер БД осуществляет поиск записей и анализирует их. Записи, удовлетворяющие условиям, могут накапливаться на сервере и после обработки запроса передаваться пользователю. Клиентское приложение может быть реализовано на языке настольных СУБД (MS Access, FoxPro, Paradox, Clipper). Взаимодействие клиентского приложения с SQL-сервером осуществляется через *ODBC-драйвер*(Open DataBase Connectivity). ODBC стал стандартом де-факто на алгоритм доступа к разнородным БД.

Достоинства архитектуры:

· снижение нагрузки на машины сервера и клиентов;

· снижение сетевого трафика и повышение эффективности обработки за счет оптимизации и буферизации ввода-вывода;

· защита данных средствами СУБД, позволяющая блокировать не разрешенные пользователю действия;

· сервер реализует управление транзакциями и может блокировать попытки одновременного изменения одних и тех же записей.

Недостатки архитектуры:

· бизнес-логика функциональной обработки и представление данных могут быть одинаковыми для нескольких клиентских приложений, что увеличивает потребности в ресурсах (повторение кода программ и запросов);

бизнес-правила функциональной обработки, сосредоточенные на клиентской части, могут быть противоречивыми.

В *архитектуре «активный сервер баз данных»* (рис. 2.10) непротиворечивость бизнес-логики контролируется на стороне сервера. Функции бизнес-логики разделяются между клиентской и серверной частями. Общие функции оформляются в виде *хранимых процедур*. Вводится механизм *триггеров*, отслеживающих события БД. При возникновении события (обычно изменения данных) выполняется оператор SQL или вызывается хранимая процедура, связанная с триггером.

Рис. 2.10. Архитектура «активный сервер баз данных»

Хранимые процедуры и триггеры могут быть использованы любыми клиентскими приложениями, работающими с БД. Это снижает дублирование программных кодов и исключает необходимость компиляции каждого запроса.

Недостатком архитектуры становится возрастающая загрузка сервера.

Такую архитектуру иногда называют *моделью с тонким клиентом,* в отличие от предыдущих архитектур, называемых *моделью с толстым клиентом*, где на стороне клиента выполняется большинство функций.

Дальнейшее снижение уровня требований к ресурсам клиента достигается за счет введения *сервера приложений,* на который переносится значительная часть программных компонентов управления данными и большая часть бизнес-логики. При этом серверы БД обеспечивают исключительно функции СУБД (рис. 2.11

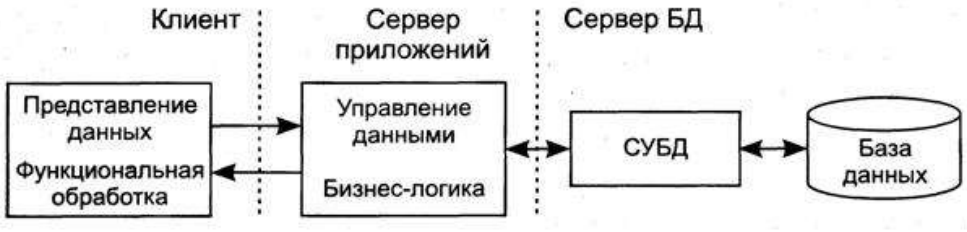

Рис. 2.11. Трехзвенная архитектура сервера приложений

Связь клиентских рабочих станций с прикладными программами на сервере приложений устанавливается через интерфейс API (Application Programming Interface). Работа клиентской части сводится к вызову функций сервера приложений.

К достоинствам трехзвенной архитектуры относятся:

· многократное использование общих функций обработки данных в множестве клиентских приложений;

· централизованное ведение бизнес-логики (при внесении изменений их не нужно тиражировать в клиентских приложениях);

· на клиентских машинах не следует устанавливать компоненту программного обеспечения управления доступом к данным;

· оптимизация доступа к БД (диспетчеризация запросов выполняется сервером приложений);

· возможность отложенного обновления БД при изменении данных в автономном режиме (данные будут обновлены в БД при следующем соединении);

· повышение скорости и надежности за счет дублирования ПО на

нескольких серверах приложений, которые могут заменять друг друга;

· перенос проверки полномочий пользователей с сервера БД на сервер приложений.

· уменьшение мощности сервера приложений за счет параллельной работы сервера приложений и сервера БД.

Тема 3. Проектирование таблиц базы данных

Основные вопросы:
3.1. Создание структуры таблицы
3.2. Индексация таблиц
3.3. Создание связей между таблицами

3.1. Создание структуры таблиц

Процесс проектирования таблиц базы данных будем излагать на примере СУБД Visual FoxPro (VFP). Проектирование системы обработки данных представляет собой непростую проблему, т. к. включает в себя очень большое число различных модулей, каждый из которых должен правильно функционировать сам по себе и взаимодействовать друг с другом.

Облегчить решение этой задачи в FoxPro призван Диспетчер проектов (Project Manager). Диспетчер проектов позволяет программисту объединить все файлы пользовательского приложения в один файл, а также не задумываться над местоположением файлов, входящих в проект. Интерфейс Диспетчера проектов позволяет иметь под руками все необходимые средства для работы с любым входящим в проект модулем. Структура Диспетчера проектов и доступные для входящих в него модулей действия понятны из внешнего вида его окна, которое показано на рис. 3.1. Сам проект хранится в таблице стандартного формата VFP, что делает его открытым для программиста. VFP сохраняет информацию о проекте в обычной таблице формата DBF, которая имеет расширение PJX, а связанный с ней файл полей примечаний имеет расширение PJT. Выбирая соответствующий элемент и нажимая кнопку New или ADD, если какой-то элемент уже создан, мы помещаем в проект новые

элементы и постепенно формируем свое будущее приложение.

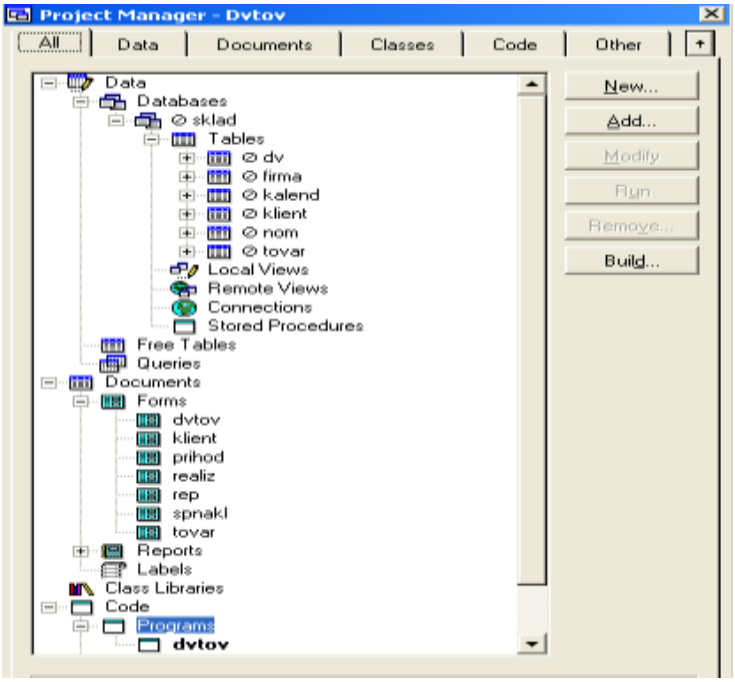

Рис. 3.1 Диспетчер проектов

Диспетчер проектов обеспечивает следующие преимущества:

– Автоматический поиск и сборку файлов, на которые есть ссылки в используемых программах.

– Автоматическое запоминание местонахождения каждого файла, объединяемого в проект прикладной программы.

– Быстрый доступ к средствам редактирования или создания каждого компонента проекта (Редактор, Конструктор отчетов и т. д.).

– Запоминание объектного кода в поле примечаний проекта, что значительно экономит место на диске.

– Автоматическая перекомпиляция исходных файлов проекта при необходимости.

– Распространение проекта путем создания на его основе исполняемого файла (APP или EXE).

Вначале создается база данных, которая в проекте может быть названа произвольно. Для последующего редактирования проекта используется команда: modify project имя проекта. Для создания таблиці БД нажмем кнопку New table для включения в базу данных новой таблицы. Появится диалоговое окно, в котором можно выбрать способ создания таблицы: либо

самостоятельно, либо с помощью Мастера создания таблиц. О Мастерах мы поговорим позже, а пока предпочтем работать самостоятельно и нажмем большую кнопку New table. В диалоговом окне Create запишем для нашей новой таблицы имя Price. После этого на экране откроется окно Конструктора таблиц. Это окно имеет три вкладки, переключение между которыми выполняется простым щелчком мыши на соответствующей вкладке:

– Fields — предназначена для описания атрибутов полей таблицы и правил работы с данными этих полей.

– Indexes — предназначена для создания или изменения индексов таблицы.

– Table — предназначена для описания правил работы с данными, хранящимися в таблице.

Конструктор таблиц, открытый на вкладке Fields, показан на рис. 3.2.

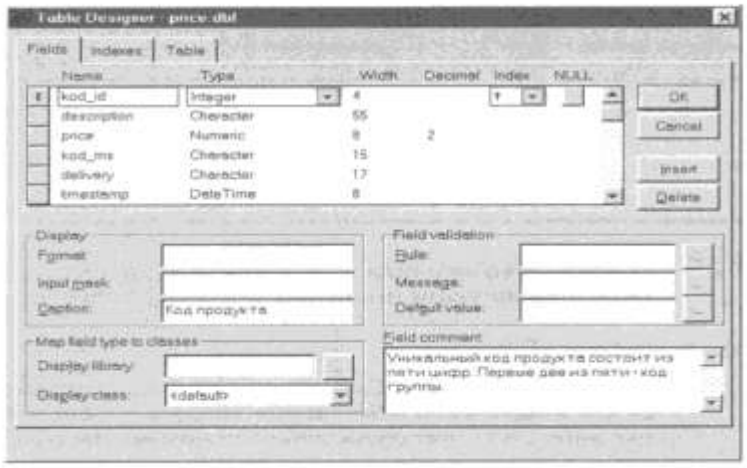

Рис. 3.2 Проектирование структуры таблицы

На вкладке Fields для каждого поля создаваемой таблицы мы должны указать:

1. В столбце Name - имя поля. Оно может включать до 128 знаков, представленных символами алфавита (как латинского, так и любого другого из поддерживаемых системой), знаков подчеркивания и цифр. Первым символом в имени поля не может быть цифра. Привилегия иметь такие длинные имена полей имеет только таблица, включенная в состав БД.

2. В столбце Type тип поля. При указании этого атрибута могут использоваться типы данных, приведенные в табл. 3.2. Для некоторых типов полей в столбцах

Width и Decimal необходимо указать длину поля и число десятичных разрядов. В этом случае в табл. 3.2 в соответствующих колонках поставлены знаки n и d.

3. В столбце Index мы можем указать необходимость создания индекса обычного типа с возрастающей или убывающей последовательностью значений. В дальнейшем можно изменить тип индекса и указать другие условия сортировки.

4. В блоке Display указываются атрибуты, связанные с режимом вывода данных поля. Об использовании формата данных (Format) и маски ввода (Input Mask) вы сможете прочитать в третьей главе при описании объектов для работы с данными. Заголовок поля Caption может очень эффективно использоваться как «дублер» имени поля везде, где требуется указать его назначение пользователю (окно Browse, форма и т. п.).

5. В блоке Field validation определяются правила проверки данных при вводе и редактировании. Например, если мы хотим ограничить наименьшее значение, которое можно ввести в поле price, числом 10, то должны указать в Rule выражение: price >= 10. Теперь при попытке ввести в поле price число, меньшее 10, будет выводиться сообщение об ошибке.

6. В блоке Field comment для полей таблицы записывается комментарий, который может пригодиться при разработке или модернизации приложения.

Таблица 3.1 Типы данных

Тип	Обозначение	Число десятичных разрядов	Объем	Описание
Character	C		1 байт на символ	Символьное выражение может содержать любые символы (до 254 для одного поля).
Currency	Y		8 байт	Денежное выражение для числовой величины. Выводит число с четырьмя десятичными разрядами и установленным обозначением используемой денежной единицы.
Date	D		8 байт	Выражение для даты может содержать день, месяц и год.
DateTime	T		8 байт	Выражение дата и время может содержать время, день, месяц и год.
Logical	L		1 байт	Булево выражение .T. или .F..

Продолжение табл. 3.1

Тип данных	Обозначение	Длина	Число десятичных разрядов	Объем	Описание
Numeric	N	n	d	от 1 до 20 байт	Числовое выражение может содержать целые или дробные числа со знаком.
Integer	I	n		4 байта	Целое число.
Double	B		d	8 байт	Числа с плавающей точкой двойной точности.
Float	F	n		от 1 до 20 байт	То же, что числовое выражение. Оставлено для совместимости.
General	G			4 байта	Поле для ссылки на объект OLE.
Memo	M			4 байта	Поле примечаний для ссылки на блок данных.
Memo	M			4 байта	Поле примечаний
(binary)					блок данных, не подвергаемых трансляции в другую кодовую страницу.

3.2. Индексация таблиц

В реляционной модели данных для связи между таблицами необходимо выделение ключевых полей. Эти же поля имеют важное значение для организации быстрого поиска данных. В СУБД информация о расположении данных в ключевых полях хранится в индексах. Чаще всего индексы располагаются в отдельных файлах и тогда говорят о создании индексных файлов. Индексы не влияют на физическое расположение данных в таблицах, а только хранят информацию о порядке расположения этих данных.

В первых версиях FoxPro в одном индексном файле мог храниться только один индекс. Этот файл имеет по умолчанию расширение IDX. Если требовалось искать данные по разным полям таблицы, приходилось открывать несколько индексных файлов и переключаться между ними. Если при редактировании данных какой-либо из индексных файлов не был открыт, информация в нем переставала соответствовать данным в таблице, и на нас обрушивался град ошибок.

В дальнейшем для решения этих проблем в FoxPro были включены составные (compound) индексные файлы, которые по умолчанию имеют

расширение CDX и могут в одном файле содержать несколько индексов. Каждый индекс в таком файле называется тегом (tag) индекса и содержит информацию о каком-либо одном ключе. Вы можете легко открыть сразу несколько тегов, открыв один составной индексный файл.

Если составной индексный файл имеет тоже самое имя, что и таблица, он называется структурным составным индексом и автоматически открывается при открытии таблицы. Когда составной индексный файл имеет имя, не совпадающее с именем таблицы, он называется независимым составным индексом. В большинстве случаев более предпочтительным оказывается использование структурного составного индекса, т. к. он обновляется автоматически при обновлении данных в таблице и тем самым требует минимальных затрат на обслуживание. Однако, если вы имеете очень большое число индексов, на их обновление могут потребоваться весьма значительные ресурсы. Выполнение программы может ощутимо замедлиться. В этом случае весьма полезно разделить все ваши индексы на две группы. В первую включить индексы, которые требуются в оперативной работе и участвуют в поиске данных и связях между таблицами, и разместить их в структурном составном индексном файле. Во вторую группу включить индексы, которые используются при составлении отчетов и могут потребоваться всего лишь несколько раз в году, и разместить их в независимом составном индексном файле. Его будет несложно привести в порядок непосредственно перед подготовкой отчета и не тратить на это время во время работы с данными.

Вернемся теперь к Конструктору таблиц и посмотрим, как создаются индексы в Visual FoxPro. С помощью вкладки Indexes Конструктора таблиц в Visual FoxPro можно создать только теги структурного составного индекса. Для создания других видов индексов необходимо использовать соответствующие команды. Если к этому моменту вы уже закрыли Конструктор таблиц с таблицей Price, то найдем ее в Диспетчере проектов. Щелкнем правой кнопкой мыши на заголовке таблицы Price и в появившемся контекстном меню выберем команду Modify. В Конструкторе таблиц перейдем на вкладку Indexes,

показанную на рис. 3.3.

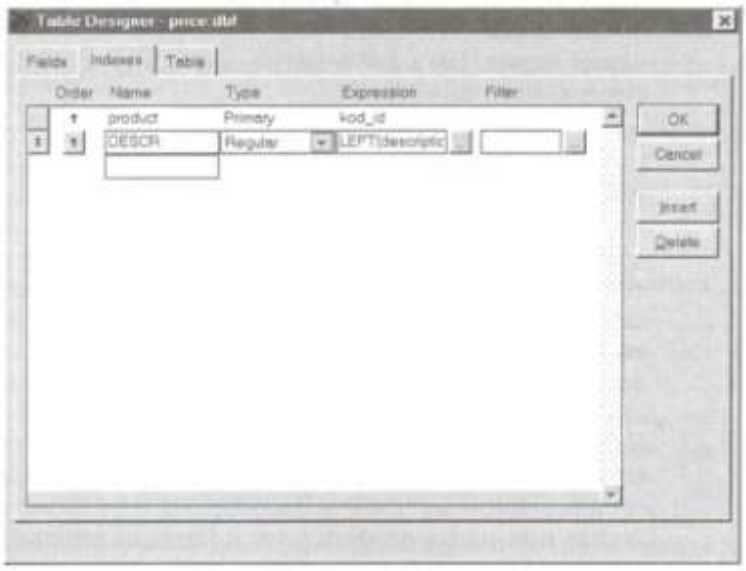

Рис. 3.3. Создание индексов

Для правильного выбора возможных опций при создании индекса приведем для них краткий комментарий.

1. В столбце Name при выборе имени тега не забудьте, что оно должно начинаться с буквы или знака подчеркивания и не может включать более 10 букв, цифр или знаков подчеркивания. Число создаваемых тегов для таблицы ограничивается только мощностью вашего компьютера, но каждый тег должен иметь уникальное имя.

2.В столбце Type мы можем выбрать один из четырех типов индекса. Если таблица включена в базу данных, вы можете создать первичный тег (primary). Этот тег в таблице может быть только один, и он не допускает наличия повторяющихся ключевых выражений. Установим первичный тег по полю kod_ce. Теперь мы не сможем для двух разных записей ввести одинаковый код продукта. Если же в этом поле до создания первичного тега уже были повторяющиеся значения, будет обнаружена ошибка, и тег не будет создан, пока мы их не устраним. Оставшиеся три типа индексов можно использовать и для таблиц, не включенных в БД.

Тег типа кандидат (candidate), так же как и первичный, не допускает наличия повторяющихся значений, но тегов этого типа в индексе может быть

несколько. Его легко при необходимости преобразовать в первичный. Уникальный тег (unique) обеспечивает вывод только первой записи из возможного множества одинаковых значений индексного выражения. Например, если вы создали уникальный индекс по полю, в котором записаны названия городов, и установили при просмотре данных этот тег управляющим, то вы не увидите городов с одинаковыми названиями. Уникальный тег очень удобно использовать для составления различных списков, например, городов, в которых находятся ваши клиенты. Тег типа кандидат не допускает дублированных значений, а уникальный тег только «отфильтровывает» такие записи. Придется смириться с некоторой путаницей, которую вносят эти не совсем удачные названия.

Регулярный тег не накладывает никаких ограничений ни на вводимые, ни на выводимые данные. Этот тип тега используется наиболее часто для поиска и управления порядком вывода данных.

3. В столбце Expression в качестве выражения для индексного ключа обычно используется имя поля, по которому создается тег. Это наиболее простой случай, и здесь не сложно в соответствующем текстовом поле набрать имя нужного поля. В некоторых случаях нам необходимо использовать ключевое выражение, которое будет включать несколько полей. Например, если мы хотим вывести список работников в алфавитном порядке, то при создании тега только по фамилии Андрей Иванов может оказаться позади Сергея Иванова. Чтобы этого не произошло, нам нужно использовать составной тег, сформированный по фамилии и имени: `lastname + firstname`

При необходимости для составления выражения можно использовать Построитель выражений, который вызывается при нажатии кнопки справа от текстового поля. Как правильно составлять выражения в Visual FoxPro, пользоваться Построителем выражений и использовать данные разного типа будет рассказано в третьей главе.

Можно использовать фильтр для того, чтобы в тег попали только некоторые записи, удовлетворяющие записанному критерию. Например, вы

можете создать тег для вывода списка клиентов только из Харькова. Для этого необходимо в текстовое поле записать соответствующее логическое выражение или воспользоваться Построителем выражений, нажав кнопку справа.

Если мы создаем тег любого типа по символьному полю, то по умолчанию данные располагаются не в алфавитном порядке, а в порядке кодов символов в ASCII-таблице. Если ваши данные записаны на английском языке и в них нет различий в использовании строчных и прописных букв, то последовательность расположения символов будет правильной. В том случае, если вы используете данные на языке, отличном от английского, возникнут расхождения с требуемой алфавитной последовательностью. За изменение этого порядка отвечает установка SET COLLATE, которая определяет требуемую последовательность символов. Наиболее часто используемые значения для этого параметра:

> GENERAL — используется для большинства западноевропейских языков.

>GERMAN — соответствует стандарту телефонного справочника в Германии.

>MACHINE — эта установка принимается по умолчанию в предыдущих версиях FoxPro.

> RUSSIAN — используется для русского языка.

В столбце Filter вы можете наложить ограничения на записи, которые будут доступны при активизации индекса. Просматривать и редактировать можно будет лишь те записи, которые будут удовлетворять указанному выражению.В столбце Order можно установить возрастающий или убывающий порядок расположения данных в индексе. В дальнейшем с помощью Конструктора таблиц вы легко можете изменить созданные индексы, удалить ненужные и добавить новые.

3.3. Создание связей между таблицами

Вернемся к проблеме создания базы данных, которую мы оставили на этапе формирования системы таблиц для хранения необходимых данных. Научившись создавать индексы, мы можем теперь установить необходимые связи между таблицами, входящими в БД. Свяжем таблицу Sales, включающую данные по продажам, с таблицей Price для того, чтобы знать, какие продукты проданы, и с таблицей Clients для того, чтобы иметь возможность быстрого определения названия фирмы-покупателя. В таблицах Price и Clients для идентификации программных продуктов и клиентов используются коды, которые имеют уникальные значения и не должны повторяться. Такие же коды используются в таблице Sales.

Убедитесь в том, что по полю kod_id создан первичный индекс для таблицы Price и таблицы Clients. Если это не так, создайте эти индексы в Конструкторе таблиц. Если вы посмотрите на таблицы Price и Clients в Конструкторе БД, то увидите, что внизу списка полей прибавилась часть под названием Indexes, и в этой части списка появилось название нового индекса с изображением ключика слева, что свидетельствует о том, что данный индекс является первичным. В таблице Sales создадим два регулярных (обычных) индекса по полям kod_id и kod_client. Для создания связей между таблицами в Конструкторе БД нажмем кнопку мыши на первичном индексе kod_id таблицы Price и, не отпуская ее, переместим указатель мыши на индекс kod_id таблицы Sales. В окне Конструктора БД мы увидим созданную связь визуально. Точно так же надо связать таблицу Clients с таблицей Sales (рис.3.4).

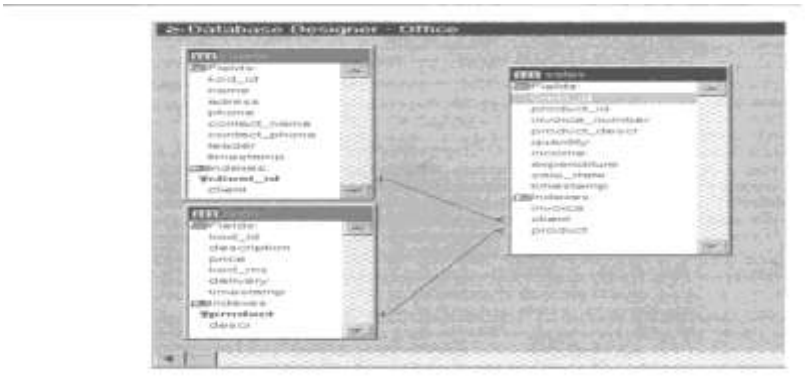

Рис. 3.4. Связи между таблицами

С.М.1.2. Проектирование баз данных

Тема 4. Программирование баз данных. Элементы объектно-ориентированного программирования для создания проектов

Для того чтобы БД были доступны для обычного пользователя необходимо создать exe-приложение. С этой целью необходимо создать проект, который позволяет проектировать формы для ввода и анализа данных, отчеты, запросы, главное меню приложения. Проектирование основано на визуальном объектно-ориентированном программировании. Следовательно, необходимо уметь использовать основные операторы и команды языка для написания кодов, которые активизируют действие СУБД после наступления того или иного события. Ниже приведены основные операторы языка xBase, который используется в VFP.

Таблица 4.1

Перейти в каталог, в котором находятся файлы проекта	**set default to c:\vfp\tovar**
Открыть проект	**modify project sklad**
Работа с таблицами базы данных	
Открыть таблицу в эксклюзивном режиме	**use имя таблицы exclusive**
Сделать активной таблицу	**select имя таблицы**
Добавить запись	**append blank**
Удалить запись	**delete**
Удалить запись по заданному условию	**delete for условие**
Полное удаление	**pack**
Просмотр всех записей	**browse**
Просмотр отдельных полей	**browse fields список полей**
Просмотр по заданному условию	**browse for условие**
Перейти в начало файла	**go top**
Перейти в конец файла	**go bottom**
Перейти на определенную запись	**go номер записи**
Найти запись по заданному условию	**locate for условие**
Перейти к следующей записи, удовлетворяющей этому же условию	**continue**
Перейти к следующей записи последовательно	**skip**
Записать в поля новой записи заданную информацию (SQL)	**insert into (список полей) values (список данных)**
Перезаписать информацию в заданном поле	**replace имя поля with информация**
Подключить индекс для сортировки данных	**set order to tag имя индекса**
Отменить подключение индекса	**set order to tag 0**
Определяет, стоит ли указатель записей на 1-ой	**EOF()**
Номер записи	**?RECNO()**
Число записей	**?RECCOUNT()**
Копирование данных из таблицы в новую таблицу	**copy to имя таблицы**
Добавление данных в таблицу из другой таблицы	**append from имя таблицы**

Продолжение табл. 4.1

Математические функции	
Преобразование строки числовых символов в число	**val(строка)**
Округление до заданного числа десятичных знаков	**round(число,кол-во десятичных знаков)**
Преобразует число в его целое значение отбрасыванием дробной части	**int(число)**
Функции работы с символьными (текстовыми выражениями)	
Определяет, содержится подстрока внутри строки	**подстрока$строка**
Определяет положение первого вхождения подстроки в строке	**AT(подстрока,строка)**
Убирает начальные и конечные пробелы	**ALLTRIM(строка)**
Преобразует дату в символьную строку	**DTOC(дата)**
Преобразует число в символьную строку	**STR(число,длина,кол-во десятичных знаков)**
Извлекает из n символов строки, начиная с позиции m	**substr(строка,m,n)**
Преобразует строчные символы в заглавные	**UPPER(строка)**
Вычисляет длину строки	**LEN(строгка)**
Заполняет поле пробелами	**space(кол-во пробелов)**
Функции работы с датами	
Преобразовать строку в дату	**CTOD(строка)**
Определяет системную дату	**DATE()**
Определяет месяц	**MONTH(дата)**
Определяет год	**YEAR(дата)**
Операторы для программирования	
Глобальные переменные и массивы	**Public ww,a(100)**

Условный оператор	**if условие**
	операторы
	else
	операторы
	endif

Выбор по условию	**iif(условие,команда1,команда2)**
Выбор в случае выполнения условий	**do case** **case условие1** **операторы** **case условие 2** **операторы** **endcase**
Оператор цикла FOR	**for i=1 to n step шаг** **операторы** **endfor**
Оператор цикла DO	**do while not eof()** **операторы** **enddo**
Оператор цикла SCAN в представленном примере информация из таблицы t1 записывается в таблицу t2	**select t1** **scan** **scatter memvar** **select t2** **append blank** **gather memvar** **endscan**
Выход из цикла	**exit**
Перейти в начало цикла	**Loop**
Активизировать отчет (просмотр и печать)	**report form имя файла отчета** **preview to print**

Продолжение табл. 4.1

Перевод наименования свойств объектов	
Заголовок	**Caption**
Фон	**BackColor**
Размер шрифта	**FontSize**
Жирный шрифт	**FontBold**
Цвет шрифта	**ForeColor**
Имя шрифта	**FontName**
Высота	**Height**
Ширина	**Width**
Подключение поля таблицы	**ControlSource**
Доступен – да (нет)	**Enabled=.t. (.f.)**
Виден на экране да (нет)	**Visible=.t. (.f.)**
Значение или величина	**Value**
Индекс (тэг)	**Tag**
Тип окна	**WindowType**
Полосы прокрутки	**ScrollBars**
Кол-во колонок	**ColumnCount**

Тема 5. Разработка проектов СУБД.

Основные вопросы:
5.1. Диспетчер проектов
5.2. Разработка форм, отчетов, меню
5.3. Создание запросов SQL SELECT

5.1. Диспетчер проектов

При создании приложения используется проект, который объединяет элементы приложения Visual FoxPro и группирует их по типам. Информация о проекте хранится в специальной таблице, которая, в отличие от обычных таблиц Visual FoxPro, имеет расширение PJX. Memo-поля таблицы содержат наименование элемента проекта, его описание и другие текстовые атрибуты. Файл с Memo-полями таблицы имеет расширение PJT. Использование проекта упрощает разработку приложения, т. к. в проекте базы данных, программы, формы, отчеты, запросы и другие элементы приложения располагаются в соответствующих разделах, а также запоминается расположение каждого включенного в проект элемента.

Создав проект и определив входящие в него элементы, вы можете использовать его для сборки приложения, построив файл с расширением APP, или для создания исполняемого файла с расширением EXE. При построении

приложения из проекта осуществляется поиск и сборка файлов, на которые ссылаются элементы приложения, отслеживаются версии файлов, входящих в проект. Вид проекта показан на рис. 3.1. В проекте имеются вкладки, которые открывают соответствующие возможности проекта:

- ALL – отражаются все файлы;
- DATA – базы данных, таблицы, запросы;
- DOCUMENTS – формы и отчеты;
- CLASSES – классы;
- CODE – программы и библиотеки;
- OTHER – меню, рисунки и прочее.

5.2. Разработка форм, отчетов, меню

Для просмотра, ввода и редактирования данных, хранящихся в таблицах, используются формы, являющиеся более наглядным средством представления информации. Рассмотрим, например, приложение, предназначенное для работы с бухгалтерскими документами, отражающими движение товаров. Естественно, что формы, предназначенные для ввода документов, должны выглядеть на экране монитора точно так же, как стандартные бланки этих документов. Другим важным преимуществом форм является то, что они позволяют работать не с одной, а с несколькими связанными таблицами, что, в свою очередь, также увеличивает наглядность.

Пользователю приложения нет необходимости знать, что такое Visual FoxPro, какие команды используются для добавления или удаления записей в таблицах. Он может даже вообще не знать, с использованием каких программных средств создавалось приложение. Для него главным является перемещение по таблице, добавление новых записей, редактирование и удаление имеющихся. Все эти возможности имеются в формах.

На рис. 5.1 показана форма, предназначенная для оформления расходной накладной. Методика конструирования форм будет изложена непосредственно при проведении лабораторных занятий по данной теме.

Любая форма связана с отчетами. Это могут быть любые печатные формы и отчеты, которые необходимы для эффективной работы на предприятии. Например, с представленной на рис. 5.1 формой связан отчет – бланк расходной накладной, показанный на рис. 5.2.

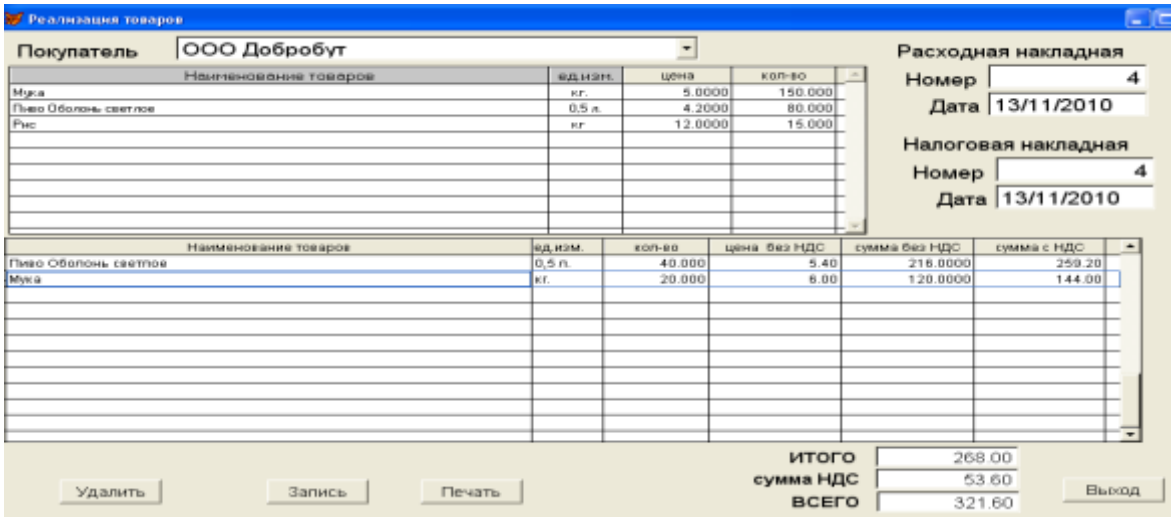

Рис. 5.1. Образец формы для оформления накладной

РАСХОДНАЯ НАКЛАДНАЯ № 4

дата 13/11/20

Покупатель ООО Добробут

Наименование товара	ед.изм.	кол-во	цена без НДС	сумма
Пиво Оболонь светлое	0,5 л.	40.000	4.2000	168.000
Мука	кг.	20.000	5.0000	100.000
			Итого	268.000
			Сумма НДС	53.600
			ВСЕГО	321.600

Рис. 5.2 – Отчет – бланк накладной

Методика конструирования отчетов будет изложена непосредственно при проведении лабораторных занятий по данной теме.

Меню приложения генерируется в рамках проекта. На рис. 5.3 показан пример проектирования главного меню с помощью средств VFP.

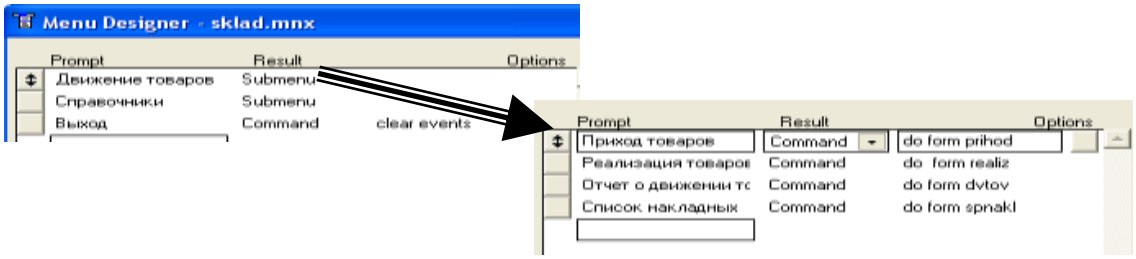

Рис. 5.3. Проектирование главного меню приложения

5.3. Создание запросов SQL SELECT

Одним из основных назначений разработанного приложения является быстрый поиск информации в базе данных и получение ответов на разнообразные вопросы. Для этих целей в Visual FoxPro используются средства, называемые *запросами.* Например, вам необходимо выбрать из таблиц информацию о клиентах, проживающих в городах, или сформировать список клиентов, купивших в последний месяц товаров на сумму свыше 5000 рублей, и упорядочить их в алфавитном порядке по полю, содержащему фамилии клиентов. Для решения таких задач предназначен конструктор запросов и команда **select** языка Visual FoxPro.

С помощью конструктора запросов Visual FoxPro вы можете формировать различной сложности критерии для выбора записей из одной или нескольких таблиц, указывая при этом, какие поля должны быть отображены в запросе. Над полями, выбираемыми из таблиц с помощью запросов, можно выполнять различные вычисления.

В отличие от традиционных команд xBase, язык SQL является множественно-ориентированным языком и направлен на получение готовых таблиц с результатами запроса. Учитывая также, что команды SQL работают с данными на уровне машинного представления, скорость получения необходимой информации возрастает в сотни раз. При этом можно уменьшить количество необходимых индексных файлов и ключей, что экономит дисковое пространство и затраты ресурсов на поддержание целостности структуры индексов. Среди современных СУБД язык SQL занимает особое положение. Навряд ли вам удастся обнаружить среди них такую, которая не позволяет использовать его возможности. При этом обратите внимание на следующие особенности:

Практически ни одна из СУБД для персональных компьютеров не поддерживает стандартную версию языка SQL. Да этим часто не могут похвастаться и серверы БД. Каждая СУБД имеет свой диалект SQL, который отличается полнотой поддержки стандарта и мелкими отличиями синтаксиса.

Из этого следует, что команда, которая поддерживается в Visual FoxPro, может отсутствовать в dBase for Windows.

В СУБД для персональных компьютеров поддержка SQL может быть выполнена по-разному. Если SQL является встроенным языком, то его команды могут переплетаться с родными командами СУБД. Так выполнена поддержка SQL в Visual FoxPro. Если SQL не является встроенным языком, то его команды должны выполняться в виде отдельных модулей.

На рис. 5.4 приведен укрупненный алгоритм построения запроса. Курсивом выделены необязательные блоки алгоритма. Приведенная логика является стандартной и отображает те необходимые элементы, из которых и будет сформирована команда для выполнения запроса. В Visual FoxPro для построения запросов в диалоговом режиме используется «Конструктор запросов» (Query Designer).

Рис. 5.4. Структура построения запроса

Формат SQL SELECT имеет вид:

SELECT *[ALL | DISTINCT] [TOP nExpression [PERCENT]] [Alias.] Select Item [AS ColumnName] [, [Alias.]Selectltem [AS ColumnName]...] FROM [FORCE] [DatabaseName!]Table [LocalAlias] [[INNER | LEFT [OUTER] \ RIGHT [OUTER] | FULL [OUTER] JOIN DatabaseName!]Table [LocalAlias] [ON JoinCondition ...] [[INTO Destination] |[TO FILE FileName [ADDITIVE] | TO PRINTER [PROMPT] | TO SCREEN]] [PREFERENCE PreferenceName] [NOCONSOLE] [PLAIN] [NOWAIT] [WHERE JoinCondition [AND JoinCondition...] [AND | ORFilterCondition[AND | ORFilterCondition...]]] [GROUP BY GroupColumn [, GroupColumn...]] [HAVING FilterCondition] [UNION [ALL] SELECTCommand] [ORDERBYOrderltem[ASC | DESC][, Orderltem[ASC | DESC]...]]*

Эта команда позволяет выполнить запрос к одной или нескольким таблицам, т. е. выбрать из них необходимые данные и направить на соответствующее устройство вывода.

Пример использования команды: запрос о движении товаров за заданный временной период:

```
select ktov,ntov,edizm,pprize, sum(iif(tip=1,iif(datan<wdata,kol,0.000),;
iif(datan<wdata,-kol,0.000))) as bkol,;
sum(iif(tip=1,iif(datan=wdata,kol,0.000),0.000)) as pkol,;
sum(iif(tip=2,iif(datan=wdata,kol,0.000),0.000)) as rkol from dv ;
 into cursor tmp  order by ntov group by 1,4
```

Список источников

1. Основная литература		
1.1.	Гайна Г.А. Основи проектування баз даних. Навч. посібник.- К.:Кондор, 2007.- 208с.	С.М.1.1-С.М.1.2
1.2	Дейт К.Дж. Введение в системы баз данных. 6-е изд., изд. Дом Вильямс, 1999.-848с.	С.М.1.1-С.М.1.2
1.3	Омельченко Л. Visual FoxPro 7.0.- СПб.:БХВ, 2002. –672 с.	С.М.1.1-С.М.1.2
1.4	Ситник Н. В. Проектування баз і сховищ даних: Навч. посібн. – К.: КНЕУ, 2004. – 348 с.	С.М.1.1-С.М.1.2
2. Дополнительная литература		
2.1.	Горев А. Visual FoxPro 5.Книга для программистов. – М.: FoxTalk, 1997 – 552 с.	С.М.1.1-С.М.1.2-
2.2	Пэддок Р., Петерсон Д. Visual FoxPro 6. Разработка корпоративных приложений. - М.: ДМК, 1999. – 552 с.	С.М.1.1-С.М.1.2-
2.3	Каратыгин С. и др. Visual FoxPro 6.- М.: ВКП, 1997. – 736 с.	С.М.1.1-С.М.1.2-
2.4	Вейскас Д. Эффективная работа с MS Access 2000- СПб.: 2000. -1040 с	С.М.1.1-С.М.1.2
	3. Методические материалы	
3.1	Методические указания к выполнению контрольной работы по курсу "Системы управления базами данных" (для студентов 3 – 4 курсов заочной формы обучения направления подготовки 6.030601 - «Менеджмент») / Харьк. нац. акад. гор. хоз-ва.; сост.: С.М Мордовцев. - Х.: ХНАГХ, 2012. - 30 с.	С.М.1.1-С.М.1.2

НАВЧАЛЬНЕ ВИДАННЯ

Конспект лекцій з курсу

«СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ»

(для студентів 3 – 4 курсів заочної форми навчання
напряму підготовки 6.030601 - «Менеджмент»)

(рос. мовою)

Укладач: **Мордовцев** Сергій Михайлович

В авторській редакції

Комп'ютерне верстання *С. М. Мордовцев*

План 2010, поз. 198Л

Підп. до друку 24.12.2010 Формат 60х84 1 /16

Друк на ризографі. Ум. друк. арк. 2,5

Зам. № Тираж 50 пр.

Видавець і виготовлювач:
Харківська національна академія міського господарства,
вул. Революції, 12, Харків, 61002
Електронна адреса: rectorat@ksame.kharkov.ua
Свідоцтво суб'єкта видавничої справи:
ДК №4064 від 12.05.2011 р.